Edícia výskumných textov
informatiky a informačných technológií

Štúdie vybraných tém softvérového inžinierstva (2)

Pokročilé metódy navrhovania programových systémov
Pokročilé metódy získavania, vyhľadávania, reprezentácie a prezentácie informácie

Kniha vznikla a bola vydaná s finančnou podporou projektu:

Projekt JPD 3 2004/1-022: Podpora vzdelávania mladých vedeckých pracovníkov s cieľom vychovať tvorivých expertov – profesionálov informatikov – pre modernú spoločnosť založenú na vedomostiach

Európsky sociálny fond pomáha rozvíjať zamestnanosť podporovaním zamestnateľnosti, obchodného ducha, rovnakých príležitostí a investovaním do ľudských zdrojov.

Mária Bieliková,
Pavol Návrat
a kol.

Štúdie vybraných tém softvérového inžinierstva 2

Pokročilé metódy navrhovania programových systémov
Pokročilé metódy získavania, vyhľadávania, reprezentácie a prezentácie informácie

Slovenská technická univerzita v Bratislave
Fakulta informatiky a informačných technológií

Edícia výskumných textov informatiky a informačných technológií

Štúdie vybraných tém softvérového inžinierstva (2)

Pokročilé metódy navrhovania programových systémov
Pokročilé metódy získavania, vyhľadávania, reprezentácie
a prezentácie informácie

Autorský kolektív:

Mária Bieliková
Pavol Návrat
Anton Andrejko
Anna Bou Ezzeddine
György Frivolt
Jaroslav Jakubík
Martin Kiselkov
Ľubomír Majtás
Ján Máté
Ivan Polický
Viliam Solčány
Martin Šechný
Miroslav Vnuk

Fakulta informatiky a informačných technológií
Slovenská technická univerzita v Bratislave
Ilkovičova 3
842 16 Bratislava

Text Design & Composition: Mária Bieliková
Copy Editor: Anton Andrejko, Martin Kiselkov
Cover Designer: Peter Kaminský

Vydala Slovenská technická univerzita v Bratislave
vo Vydavateľstve STU, Bratislava, Vazovova 5.

ISBN

PREDHOVOR

Publikácia, ktorá sa vám dostala do rúk, je už druhou v poradí, ktoré sme v relatívne krátkom čase dokázali pripraviť v *Edícii výskumných textov informatiky a informačných technológií* na témy z oblasti programových a informačných systémov. Obe *Štúdie vybraných tém softvérového inžinierstva* sa venujú sa dvom ťažiskovým okruhom. Prvým okruhom sú pokročilé metódy navrhovania programových systémov. Druhým okruhom sú pokročilé metódy získavania, vyhľadávania, reprezentácie a prezentácie informácií. Voľba oboch okruhov tém nebola náhodná. Všetky témy sú aktuálnymi témami súčasného výskumu v oblasti programových a informačných systémov. Ako také sú predmetom záujmu a štúdia výskumných študentov, t.j. najmä študentov doktorandského štúdia. Oni sú nielen prvými čitateľmi *Štúdií*, vybraní doktorandi sú aj autormi jednotlivých častí v oboch publikáciách.

Prvé *Štúdie vybraných tém softvérového inžinierstva* sa v rámci uvedených okruhov zaoberali dvomi ťažiskovými témami. Prvou témou bola analýza návrhových vzorov, ktoré predstavujú jednu z kľúčových oblastí vyvíjajúcej sa disciplíny softvérového inžinierstva. Druhá časť obsahovala päť štúdií z vybraných tém programových a informačných systémov, ktoré diskutujú a analyzujú otvorené vedecké problémy v predmetnej oblasti aj v spojitosti so spracovaním informácií na internete.

Obdobný postup ako pri prvých *Štúdiách* sme zvolili aj pri tomto zväzku. Vznikol na základe seminárov študentov doktorandského štúdia študijného programu programové systémy v odbore softvérové inžinierstvo. Semináre podporil projekt Európskych štrukturálnych fondov, ktorého hlavným cieľom je podpora vzdelávania prostredníctvom motivačných nástrojov pre doktorandov a zvyšovaním kvality vzdelávania v treťom stupni vysokoškolského štúdia v oblasti informatiky a informačných technológií.

Informatika a informačné technológie sú kľúčovým prvkom budovania modernej spoločnosti „založenej na vedomostiach", ako je dnes módne vravieť. Mladí talentovaní absolventi druhého stupňa vysokoškolského štúdia v oblasti informatiky alebo príbuzných oblastiach majú v súčasnosti veľké možnosti uplatnenia sa v praxi. Informačná spoločnosť však potrebuje aj špecializovaných odborníkov a vedeckých pracovníkov s ukončeným tretím stupňom vysokoškolského štúdia v študijných odboroch skupiny informatických vied, informačných a komunikačných technológií tak, aby bolo možné budovať ekonomiku založenú na najnovších vedeckých poznatkoch. V širšom kontexte ide o rozvoj spoločnosti (ak chcete, založenej na vedomostiach), nielen ekonomiky, schopnej vyrovnávať sa so zložitými výzvami, ktoré pred ňou stoja. S tým súvisí potreba profesionálov v oblasti uchovávania, spracúvania a prezentácie informácií v bohatej palete reprezentácií ako základného prvku informačnej spoločnosti.

S rozvojom informatiky a informačných technológií a s posunom spoločnosti k informačnej spoločnosti, resp. spoločnosti založenej na vedomostiach, vzniká potreba

vychovávať odborníkov v špecializovaných oblastiach. Semináre, ktoré sa uskutočňujú na Fakulte informatiky a informačných technológií Slovenskej technickej univerzity v Bratislave v rámci doktorandského štúdia a podporené projektom, sa zameriavajú na rôzne oblasti programových a informačných systémov. Zatiaľ čo v prvom zväzku *Štúdií* sme podchytili seminár venovaný návrhovým vzorom, v tomto zväzku sme spracovali témy seminára, venovaného webovej inteligencii. Je to veľmi nová oblasť, možno bližšia viac informačným systémom, avšak v každom prípade potenciálne veľmi užitočná pre každého študenta softvérového inžinierstva..

Našou ambíciou bolo sprístupniť záujemcom o softvérové inžinierstvo vybrané témy a tým zdieľať výsledky seminárov a tvorivého prístupu študentov k jednotlivým témam v rámci diskusií. Výskumné texty v tejto publikácii sú vhodné aj pre študentov ďalších študijných programov v odboroch ako napr. informatika, aplikovaná informatika, informačné systémy, či umelá inteligencia a to v študijných programoch uskutočňovaných na Slovenskej technickej univerzite v Bratislave a aj na iných univerzitách.

Publikácia pozostáva z dvoch dielov. V prvom (Diel 1: Webová inteligencia) sa sústreďujeme na analýzu rôznych aspektov toho, čo sa začalo nazývať webová inteligencia. Druhý (Diel 2: Vybrané témy programových a informačných systémov) obsahuje štyri štúdie, ktoré diskutujú a analyzujú vybrané otvorené vedecké problémy z dynamicky sa rozvíjajúcej oblasti programových systémov so špeciálnym dôrazom na programové informačné systémy aj v spojitosti s internetom.

Diel 1: Webová inteligencia

Čo je to webová inteligencia? Ide o nový pojem. Jeho obsah sa vytvára práve prebiehajúcim výskumom. Výskum sa dotýka sa širokého spektra otázok, súvisiacich s vývojom ďalšej generácie webu. Je to nový smer vo výskume a vývoji, v ktorom sa skúma, akú rolu alebo praktické dôsledky môže zohrať umelá inteligencia a pokročilé informačné technológie vo vývoji systémov, služieb a prostredí opierajúcich sa o web. Takto to aspoň vidia Ning Zhong, Jiming Liu a Yiyu Yao, ktorí sa podujali zostaviť vôbec prvú (viacautorskú) vedeckú monografiu na túto tému, ktorú vydal v roku 2003 Springer. Knižka sa snaží poskytnúť podrobný úvod a systematický prehľad tejto novej oblasti výskumu. Ponúka štúdie súčasného stavu výskumu jednotlivých problémov. Zaoberá sa tiež niektorými aplikačnými aspektami.

Práve toto boli hlavné dôvody, pre ktoré sme sa rozhodli zamerať doktorandský seminár na jar 2006 na webovú inteligenciu v takom chápaní, v akom ho prezentuje uvedená knižka. Vybrané kapitoly sa stali základom pre referáty, ktoré boli úvodmi pre seminárne diskusie. Seminár v rámci doktorandského štúdia viedol Pavol Návrat. Doktorandi, ktorí referáty predniesli, dopracovali ich textovú podobu potom do výsledného tvaru, ktorý máme možnosť čítať v tomto zväzku.

Každá kapitola je tak výsledkom tvorivej činnosti, ku ktorej prispeli viacerí. Samotný text každej z nich písal ten-ktorý doktorand a jeho autorský prínos treba čo najvýraznejšie zdôrazniť a oceniť. Na seminároch prebiehala diskusia, na ktorej sa zúčastňovala celá skupina doktorandov a ktorá v tom-ktorom prípade ovplyvnila definitívne znenie opisu. Napriek tomu považujeme za korektné, aby sme označili ako jediných autorov jednotlivých opisov doktorandov, ktorí im dali písomnú podobu.

Náš výber tém z webovej inteligencie, ktorý sme zaradili do seminára (a teda aj do tejto knižky), možno rozčleniť do piatich okruhov (kapitol tejto publikácie): webové

agenty (2 témy), dolovanie v dátach na webe (4 témy), vyhľadávanie informácií na webe a manažment znalostí (4 témy), infraštruktúra pre webové inteligentné systémy (2 témy) a inteligencia sociálnych sietí (2 témy).

Autori sa podieľali na jednotlivých kapitolách takto:

- *Webové agenty:*
  - Agentové služby v DAML-S: Martin Kiselkov
  - Návrh scenárov pre sociálne agenty: Ľubomír Majtás
- *Dolovanie v dátach na webe:*
  - Vyhľadávanie informácií porovnávaním webových stránok: Miroslav Vnuk
  - Zisťovanie nepriamych asociácií z dát o používaní webu: Anna Bou Ezzedine
  - Znalostná indukcia obaľovačov určených na získavanie informácií z webu: Ľubomír Majtás
  - Dolovanie webových záznamov: Ján Máté
- *Vyhľadávanie informácií na webe a manažment znalostí:*
  - Osobné a zamerané webové pavúky: Martin Kiselkov
  - Reprezentácia, zdieľanie a získavanie znalostí na webe: Miroslav Vnuk
  - Manažment znalostí na webe so sémantikou: Martin Šechný
  - Ontológia – objavovanie taxonomických relácií z webu: Anna Bou Ezzedine
- *Infraštruktúra pre webové inteligentné systémy:*
  - Algoritmické aspekty webových inteligentných systémov: Ivan Polický
  - Predpríprava webových dokumentov na Internete: Ján Máté
- *Inteligencia sociálnych sietí:*
  - Sociálne (spoločenské) siete – od webu k manažmentu znalostí: Martin Šechný
  - Algoritmus usporiadania na určenie miery dobrej povesti alebo relevancie založený na topológii grafu: Ivan Polický

Diel 2: Vybrané témy programových a informačných systémov

Do druhej časti zaraďujeme štyri štúdie, ktoré sa venujú vybraným otvoreným vedeckým problémom, týkajúcim sa programových a informačných systémov. Ide o oblasti, v ktorých prebieha veľmi intenzívny vývoj. Programové systémy sa stávajú systémami, pôsobiacimi v čoraz rôznorodejšom prostredí, vrátane internetu. Stávajú sa súčasťou čoraz komplexnejších systémov – na jednej strane rozsiahlych informačných systémov, na druhej strane systémov, spolu určených technickou platformou, ktorou už dávno nie je len počítač v klasickom slova zmysle, ale aj najrôznejšie vnorené systémy, (tele)komunikačné systémy a pod.

Informačné systémy sa stávajú univerzálnym modelom spôsobov vyhľadávania, získavania, sprístupňovania, uchovávania, odovzdávania, spoločného používania, prezentovania informácií. I keď sa v zásade dá na ne nazerať odhliadnuc od toho, či sú operácie a procesy podporené počítačom alebo nie, čoraz viac sa zväčšuje praktický význam informačných systémov, ktoré sú realizované pomocou programových systémov (a tie samozrejme pomocou počítačových systémov alebo iných technických systémov, zahŕňajúcich počítače). Je to najmä preto, že softvérovo podporené infor-

mačné systémy majú vďaka možnostiam, ktoré poskytuje naprogramovaný počítač, výhody, ktoré sa ručným spracovaním nedajú dosiahnuť. Toto je súčasne aj argumentom pre úzke prepojenie výskumu v oboch oblastiach – ako softvérového inžinierstva, tak aj informačných systémov.

Štúdie sú výsledkom práce doktorandov v rámci ich doktorandského štúdia. Možno nezaškodí pripomenúť, že doktorandské štúdium sa koná pod vedením školiteľa. Na každej štúdii má preto podiel aj príslušný školiteľ. Napriek tomu však považujeme za korektné, aby sme označili ako jediných autorov jednotlivých štúdií doktorandov, ktorí im dali písomnú podobu a ktorí ich predložili a úspešne obhájili ako písomnú časť svojej dizertačnej skúšky.

Autori sa podieľali na jednotlivých kapitolách takto:

- Modelovanie používateľa v aplikáciách opierajúcich sa o web: Anton Andrejko (školiteľ prof. Mária Bieliková)
- Paralelná a distribuovaná simulácia systémov s diskrétnymi udalosťami: Viliam Solčány (školiteľ: prof. Jiří Šafařík)
- Štruktúry komunít v sieťach: György Frivolt (školiteľ prof. Mária Bieliková)
- Znovupoužitie návrhových vzorov na úrovni zdrojového kódu: Jaroslav Jakubík (školiteľ prof. Pavol Návrat)

Dúfame, že táto knižka poslúži záujemcom o poznanie programových a informačných systémov. Umožňuje spoločne využiť výsledky štúdia v tejto oblasti. Tešíme sa na prípadné odozvy alebo pripomienky.

November 2006,
Bratislava

Mária Bieliková a Pavol Návrat

OBSAH

DIEL I: WEBOVÁ INTELIGENCIA 1

1 WEBOVÉ AGENTY 5
1.1 Agentové služby v DAML-S 5
1.2 Návrh scenárov pre sociálne agenty 11

2 DOLOVANIE V DÁTACH NA WEBE 19
2.1 Vyhľadávanie informácií porovnávaním webových stránok 19
2.2 Zisťovanie nepriamych asociácií z dát o používaní webu 24
2.3 Znalostná indukcia obaľovačov určených na získavanie informácií z webu 30
2.4 Dolovanie vo webových záznamoch 38

3 VYHĽADÁVANIE INFORMÁCIÍ NA WEBE A MANAŽMENT ZNALOSTÍ 45
3.1 Osobné a zamerané webové pavúky 45
3.2 Reprezentácia, zdieľanie a získavanie znalostí na webe 50
3.3 Manažment znalostí na webe so sémantikou 55
3.4 Ontológia – objavovanie taxonomických relácií z webu 60

4 INFRAŠTRUKTÚRA PRE WEBOVÉ INTELIGENTNÉ SYSTÉMY 67
4.1 Algoritmické aspekty webových inteligentných systémov 67
4.2 Predpríprava webových dokumentov na Internete 74

5 INTELIGENCIA SOCIÁLNYCH SIETÍ 81
5.1 Sociálne (spoločenské) siete – od webu k manažmentu znalostí 81
5.2 Algoritmus usporiadania na určenie miery dobrej povesti alebo relevancie založený na topológii grafu 85

POUŽITÁ LITERATÚRA 93

DIEL II: VYBRANÉ TÉMY PROGRAMOVÝCH A INFORMAČNÝCH SYSTÉMOV 97

6 MODELING THE USER IN THE WEB-BASED APPLICATIONS 99
6.1 Approaches to user modeling 99
6.2 User characteristics and sources for user modeling 104
6.3 Semantic Web environment 107
6.4 User model representation 111
6.5 Automatic user modeling 118
6.6 Lifetime and maintenance of user models 121
References 122

7 PARALLEL AND DISTRIBUTED SIMULATION OF DISCRETE EVENT SYSTEMS 127
7.1 Discrete event simulation 128
7.2 Parallelizing discrete event simulation 132
7.3 The issue of synchronization 134
7.4 The lookahead 146
7.5 Discrepancies between DES and PDES 154
7.6 High level architecture 159
7.7 Conclusion 162
References 162

8 COMMUNITY STRUCTURES IN NETWORKS 167
8.1 Models of networks 167
8.2 Network clustering techniques 173
References 182

9 ZNOVUPOUŽITIE NÁVRHOVÝCH VZOROV NA ÚROVNI ZDROJOVÉHO KÓDU 185
9.1 Znovupoužitie vzorov 186
9.2 Dekompozícia vzoru 189
9.3 Využitie CASE nástrojov pre dekompozíciu vzoru 194
9.4 Využitie atypických jazykov pre dekompozíciu vzoru 200
9.5 Použitie štandardných jazykov pre dekompozíciu vzoru 210
9.6 Rozšírenie množiny prostriedkov jazyka JAVA 213
9.7 Zhodnotenie 222
Použitá literatúra 224

DIEL I:
WEBOVÁ INTELIGENCIA

Východisková literatúra

V tejto časti prezentujeme štúdie vytvorené na základe vybraných častí publikácie

> ZHONG, N. – LIU, J. – YAO, Y. (Eds.) (2003) *Web Intelligence*. 1st edition. Springer, pp. 440. ISBN-10: 3-540-44384-3

Konkrétne sa venujeme týmto témam:

- webové agenty,
- dolovanie v dátach na webe,
- vyhľadávanie informácií na webe a manažment znalostí,
- infraštruktúra pre webové inteligentné systémy,
- inteligencia sociálnych sietí.

Pri ich rozpracovaní sme použili aj ďalšie zdroje, ktoré sa nachádzajú v samostatnej časti na konci tohto dielu.

1 WEBOVÉ AGENTY

Agenty sú relatívne autonómne systémy, ktorých účelom je na základe akcií splniť určité ciele vo vybranom prostredí. Webové agenty, ako názov ich názov predurčuje, sú agenty pracujúce v prostredí webu, či už na internete alebo v rámci rôznych intranetov. Ide o komplexné softvérové systémy určené na vykonávanie mnohých úloh, od smerovania, vyhľadávania, kategorizovania, filtrovania až po prezentovanie informácií. Jednoduchým príkladom webového agenta je prehliadač webových stránok. Od používateľa získa adresu zdroja informácií, ku ktorému sa pripojí, získa z neho tieto informácie a prezentuje ich používateľovi. Iným príkladom môžu byť agenty rôznych vyhľadávacích portálov, ktoré cyklicky prechádzajú webové stránky celého internetu a ich obsah indexujú do databáz. Následne, keď používateľ zadá kľúčové slová do vyhľadávača, ten len vyčíta informácie z databázy, bez toho aby reálne prehľadával web.

Nasledujúca časť ukazuje dve rôzne aplikácie agentov. Prvá sa týka aplikácie agentov v prostredí webu so sémantikou, druhá opisuje sociálne agenty, ktorých úlohou je riešiť interakciu používateľa s počítačom.

1.1 Agentové služby v DAML-S

World Wide Web, ktorý bol navrhnutý ako jednoduchý komunikačný systém pre výmenu hypertextových stránok, sa stal v poslednom čase veľmi úspešným systémom a čoraz viac sa používa nielen pre zobrazovanie statických informácií, ale aj ako používateľské rozhranie pre prístup k webovým informačným službám. Tieto služby pokrývajú široké spektrum oblastí od elektronického obchodu, cez vyhľadávacie a encyklopedické systémy až po počítačové hry. Ich spoločnou charakteristikou okrem toho, že sú prístupné cez web, je ich náročné automatizované spracovanie [1]. Komunikácia a interakcia s nimi je navrhnutá pre ľudí, nie pre počítače.

Jeden zo smerov útoku na tento veľký problém je web so sémantikou (Antoniou & van Harmelen, 2004), ktorý sa snaží o zmenu webu tak, aby bol prístupný pomocou techník umelej inteligencie. DAML-S (Ankolekar et al., 2002) je ontológiou opísanou v jazyku DAML+OIL[2], navrhnutou pre opis webových služieb. Článok (Bryson et al., 2003) obsahuje prehľad aktuálneho stavu prác na DAML-S a softvérového inžinierstva založeného na agentoch. Poskytuje základy tvorby inteligentných agentov, pričom sa

[1] Niektoré webové služby, ako napr. výmena súborov pomocou služby Rapidshare, poštová služba Gmail, sú zámerne navrhnuté tak, aby sa sťažilo ich automatizované využívanie. Web stránky týchto služieb zobrazujú okrem užitočných informácií aj platenú reklamu a pokiaľ by sa dali tieto služby používať automatizovane, bez toho, aby si používateľ musel pozrieť reklamu, tak by prevádzkovatelia takýchto služieb stratili jednu z možností financovania.

[2] The DARPA Agent Markup Language, http://www.daml.org/

zameriava na modulárne, reaktívne prístupy návrhu agenta, lebo tieto sú vhodné pre vysoko distribuovanú, zložitú povahu webovej inteligencie. Taktiež obsahuje návrh na rozšírenie DAML-S. Motivácia, prečo použiť formalizmus DAML-S procesnej ontológie je tá, že umožňuje, aby agent, ktorý uvažuje o svojom dočasnom rozšírení pomocou webovej služby mohol použiť odvodzovanie a kontrolu dôkazov.

1.1.1 Základné pojmy

Základné pojmy, ktoré článok (Bryson et al., 2003) používa, sú:

- Webová služba – program alebo zariadenie, ktoré je prístupné cez web.
- Zložená služba – služba, ktorá kombinuje preddefinované čiastkové služby spôsobom výhodným pre používateľa. Pomocou hierarchie služieb, ako napr. v DAML-S, nie je potrebné presne špecifikovať čiastkové služby, aby mohli byť skombinované, stačí iba špecifikovať požadované vlastnosti.
- Agent – relatívne autonómny systém s množinami:
  - cieľov – podmienky, ktoré sa snaží agent splniť,
  - zámerov – ciele a podciele, ktoré agent v súčasnosti vykonáva,
  - presvedčení – znalosti agenta o svete (ktoré sú nutne ohraničené a pravdepodobne nepresné),
  - správaní – akcie, ktoré je agent schopný vykonať.

Vo všeobecnosti sa na agenty pozerá ako na používateľov služieb, ale v článku sa zdôrazňuje, že sa na výsledky služieb dá pozerať aj ako na akcie, ktoré môže použiť agent pre dosiahnutie svojich cieľov.

1.1.2 Služby webu so sémantikou

Web so sémantikou sa nemusí obmedzovať iba na jednoduché objavovanie a dotazovanie informácií, má aj veľký potenciál v automatizácii služieb. DAML-S bol navrhnutý pre široké spektrum interakcií a jeho hlavné zameranie je na webové stránky, ktoré idú ďalej v možnostiach webu, ako iba poskytovanie statických informácií.

DAML-S v súčasnosti opisuje služby v troch konceptuálnych oblastiach: profil, procesný model a základy. Rozoberieme ich dôkladnejšie:

- Profil – opisuje, čo služba robí. Charakterizuje službu pre účely vyhľadávania a vyberania, t.j. poskytuje informácie, aby agent mohol posúdiť, či mu služba vyhovuje. Keďže na zápis profilu sa používa deskripčná logika z DAML+OIL, profily sú rozdelené do klasifikačných schém. Pre všetky triedy služieb môže profil opisovať vstupy, výstupy, štartovacie podmienky a dôsledky. Konkrétne triedy profilov môžu obsahovať ďalšie charakteristiky.
- Procesný model – opisuje, ako služba funguje. V ňom sú zahrnuté informácie o vstupoch služby (spolu s príznakmi o tom, či sú povinné alebo nepovinné), výstupoch služby (možnou súčasťou je aj opis podmienok, za ktorých sa môžu rôzne výstupy vyskytnúť), štartovacie podmienky (ktoré musia byť splnené, aby sa služba dala použiť) a dôsledky (ktoré vzniknú v dôsledku použitia služby). Pre zložité služby procesný model opisuje, ako sa dajú rozložiť na jednoduchšie procesy a aký je medzi nimi tok riadenia.

- Základy – opisujú, ako sa služba používa, t.j. definuje, ako môže agent pristupovať ku službe. Väčšinou je tu definovaný prenosový protokol (napr. RPC, HTTP-FORM, CORBA, vzdialené volania), parametre protokolu (napr. čísla portov) a kódovanie dátových elementov pri prenose.

1.1.3 DAML-S procesy

Procesný model DAML-S navrhli pre širokú škálu služieb a čerpá pritom z výskumu plánovacích jazykov (McDermott et al., 1998), práce na programovacích jazykoch a distribuovaných systémoch.

Vstupy, výstupy, štartovacie podmienky a dôsledky sú najdôležitejšou charakteristikou procesu. Vstupy a výstupy sú pomenované a typované pomocou DAML+OIL tried alebo primitívnych dátových typov z XML Schema štandardu. DAML-S obsahuje tri typy procesov: atomické, jednoduché a zložené.

- Atomické procesy sú najmenšie opisované jednotky, sú analogické volaniu procedúr v programovacích jazykoch. Atomické procesy sú, z pohľadu používateľa služby, vykonávané v jednom kroku.
- Jednoduché procesy sú ponímané podobne ako atomické procesy, majú jednokrokové vykonávanie. Na rozdiel od atomických procesov ale nie sú priamo vykonateľné a nie sú pre ne definované základy. Používajú sa na vytvorenie abstraktného pohľadu na atomické procesy alebo zložené procesy.
- Zložené procesy sú poskladané z podprocesov, ktoré môžu byť atomické, jednoduché alebo zložené. Na ich spájanie sa používajú riadiace konštrukcie, sú opísané v tabuľke 1-1.

Tabuľka 1-1. Riadiace konštrukcie v DAML-S.

Konštrukcia	**Význam**
Sequence	Vykonaj sekvenčne postupnosť procesov
Concurrent	Vykonaj množinu procesov súbežne
Split	Vykonaj prvky množiny procesov
Split+Join	Vykonaj prvky množiny procesov a zosynchronizuj ich
Choice	Rozhodni sa medzi alternatívami
If-Then-Else	Ak je splnená definovaná podmienka, tak vykonaj „Then“, inak „Else“
Repeat-Until	Cyklicky vykonaj množinu procesov, až kým sa nesplní podmienka
Repeat-While	Cyklicky vykonaj množinu procesov, ak platí podmienka
Unordered	Vykonaj všetky procesy z množiny v ľubovoľnom poradí

1.1.4 Agenty a programovanie pre web so sémantikou

Článok (Bryson et al., 2003) sa taktiež venuje vývoju softvéru pre web so sémantikou. Poukazuje na to, že agentovo orientovaný prístup k softvérovému inžinierstvu je možným riešením problémov vývoja webu so sémantikou. Ďalej opisuje dve základné otázky, ktoré treba vyriešiť pre komplexných vývoj agentov: otázka modularity a otázka výberu akcií.

Úvahy autorov vychádzajú zo softvérovej metodológie nazývanej „návrh orientovaný na správanie“ (angl. Behavior Oriented Design – BOD) (Bryson & Stein, 2001). BOD je jedným zo v súčasnosti prevažujúcich prístupov k návrhu agentov: je hybridom medzi modulárnymi systémami založenými na správaní a reaktívnom plánovaní. BOD agenty sa skladajú prevažne z viacerých modulov, ktoré priamo ovládajú celé správanie agenta (akcie, vnímanie, učenie). BOD na rozdiel od iných architektúr kladie dôraz na silu a autonómnosť modulov a oslabuje význam plánov pre rozhodovanie medzi modulmi.

Modularita

Modularita je dôležitá technika pri zjednodušovaní softvéru. Zložitý program môže byť rozdelený na množinu relatívne jednoduchých modulov, ktoré môžu byť vyvíjané a testované nezávisle na sebe. Pri rozklade ale vznikajú tieto otázky:

- Koľko modulov by malo existovať?
- Čo bude v jednotlivých moduloch?

Našťastie sa na tieto otázky dajú aplikovať riešenia vyvinuté v rámci objektovo-orientovaného návrhu (angl. object-oriented design). Objektovo-orientovaný návrh odporúča, aby sa program rozdelil podľa toho, ako sa musí manipulovať s premenlivým stavom. Stav je srdcom objektu, zatiaľ čo jeho metódy sú akciami nad týmto stavom.

Jednou z myšlienok BOD je to, že podobné uvažovanie sa dá aplikovať aj na umelé agenty. Hlavným kritériom pre posudzovanie modulu správania sú jeho akcie a tieto akcie musia byť podporované vnímaním a pamäťou. Vnímanie kombinuje vstupy agenta s jeho znalosťami a presvedčeniami.

Výber akcií

Hlavná výhoda modulárnej inteligencie je relatívna jednoduchosť jej jednotiek. Nevýhodou rozdelenia inteligencie do modulov je to, že rozličné moduly môžu vykonávať protichodné akcie, napríklad môže vzniknúť konflikt pri používaní obmedzených prostriedkov.

Prvotné modulárne systémy umelej inteligencie odmietali systém centrálneho riešenia konfliktov, lebo toto riešenie bolo považované za nedostatočné a znovu vytváralo problém, ktorý sa modularizácia snažila riešiť (Maes, 1990). Namiesto toho sa očakávalo od modulov správania, aby samé dokázali posúdiť podmienky svojej použiteľnosti. Na nešťastie tento prístup viedol ku kombinatorickým problémom v komplexných agentoch, kde mohli byť viaceré správania vykonané v podobných kontextoch a to si vyžadovalo, aby každé správanie modelovalo aj iné správania. Neskôr sa ukázalo, že rozhodovanie medzi modulmi je menej zložité ako monolitické smery inteligencie.

V súčasnosti prevažujúci spôsob, ako rozhodovať v modulárnych systémoch založených na správaní, je pomocou hierarchických reaktívnych plánov. Reaktívne pláno-

vanie vyberá akcie vždy podľa aktívneho kontextu na rozdiel od konštruktívneho plánovania, ktoré zahŕňa proces prehľadávania. Hierarchické reaktívne plány sú jednoduché robustné plány, kde každý element môže byť sám ďalším reaktívnym plánom.

BOD obsahuje špecifikáciu pre paralelne začínajúce zásobnikovo usporiadané hierarchické plány reaktívne plány (angl. parallel-rooted, ordered slip-stack hierachical (POSH) reactive plans) (Bryson & Stein, 2001). Článok sa zameriava na dva relatívne všeobecné druhy reaktívny plánov, jednoduché postupnosti a základné reaktívne plány (angl. basic reactive plan – BRP).

Základné reaktívne plány

Tieto plány boli použité vo viacerých architektúrach (Fikes et al., 1993, Nillson, 1994) a sú charakteristické pre reaktívne plánovanie.

BRP krok je trojica (π, ρ, α), kde π je priorita, ρ je spúšťač a α je akcia. *BRP* je malá množina (väčšinou 3-7 prvková) krokov plánu $\{(\pi_i, \rho_i, \alpha_i)^*\}$ zviazaná s dosiahnutím príslušného cieľa. Spúšťač ρ_i je je konjunkciou boolovských primitív, ktoré určujú vykonateľnosť príslušného kroku. Každá priorita π_i je prvkom lineárne usporiadanej množiny. Každá akcia α_i je môže byť iným BRP alebo primitívnou akciou.

Poradie vykonávania plánovaných krokov sa určuje dvoma spôsobmi: prioritou a spúšťačmi. Keď sa môže vykonať viac ako jeden krok, tak sa vykoná krok s najvyššou prioritou. Pokiaľ sa môžu vykonať viaceré kroky s rovnakou prioritou, tak sa vykoná ľubovoľný jeden z nich. Normálne sa vykonávania pre kroky s rovnakou prioritou navzájom vylučujú. Pokiaľ sa nedá vykonať žiadny krok tak sa BRP skončí. Cieľový krok má väčšinou najvyššou prioritu, v tomto prípade jeho spúšťač kontroluje, či bol BRP úspešný a jeho akcia ukončuje BRP.

Tabuľka 1-2. Základný reaktívny plán pre simulovaného hráča.

Krok	**Priorita**	**Spúštač → Akcia**
F	4	lopta v bránke → gól
E	3	mám loptu & som pri bránke → strieľaj
D	2	mám loptu & spoluhráč vo výhodnej polohe → prihraj
C	1	mám loptu → bež dopredu
B	1	nemám loptu & som blízko lopty → zober loptu
A	0	*true* → čakaj

Napríklad základný reaktívny plán pre hráča v simulovanom počítačovom futbale Robocup by mohol vyzerať tak, ako v tabuľke 1-2. Keď sa hráč začne vykonávať a lopta je už v bránke, tak nemá zmysel vykonávať žiadne akcie, lebo bol gól. Preto je krok *F* cieľový. V opačnom prípade, pokiaľ má hráč loptu, tak môže alebo strieľať na bránku *E* alebo prihrať spoluhráčovi *D*, alebo bežať s loptou *C*, priority zodpovedajú tomu, že je najvýhodnejšie priamo vystreliť na bránku (pokiaľ sa to dá). Pokiaľ sa to nedá, tak prihrať spoluhráčovi (pokiaľ je vo výhodnejšej polohe), alebo ísť ďalej s loptou (ale to unavuje hráča viac ako streľba na bránku alebo prihrávka). Úmerne k výhodnosti pre hráča sú krokom priradené rôzne priority. Pokiaľ hráč nemá loptu pod kontrolou, tak sa môže pokúsiť ju ovládnuť *B*. Pokiaľ hráč nevie urobiť nič, tak čaká *A*.

Tento plán nie je zďaleka úplný, vôbec nezahŕňa obranu, presun hráča na výhodné pozície, keď nemá loptu, spoluprácu medzi hráčmi a ďalšie aspekty hry. Taktiež netriviálne akcie (ako napríklad zobratie lopty alebo beh s loptou) sa musia rozdeliť na jednoduchšie akcie a tie sú potom riadené plánmi na nižších úrovniach.

Najväčšou výhodou BRP je to, že sa dajú relatívne jednoducho vytvárať. Programátorovi stačí, keď si predstaví najhorší možný prípad pri riešení daného cieľa a súčasne ignoruje nadbytočné kroky. Priority pre každý krok sú potom nastavené v opačnom poradí, ako by sa kroky mohli vykonávať. Následne sa doplnia spúšťače, pričom sa začína krokom s najvyššou prioritou.

1.1.5 Webové služby ako správanie agentov

Článok (Bryson et al., 2003) udáva tri dôvody, prečo je výhodné sa pozerať na webové služby ako na agenty:

1. Webové služby sú analogické správaniu modulov.
2. Veľká časť výskumu urobeného pre modulárnu inteligenciu by sa dala použiť pre vývoj zložených služieb.
3. Programovacie techniky agentovo orientovaného softvérového inžinierstva by boli užitočné pri vývoji webu so sémantikou.

1.1.6 Rozšírenia pre DAML-S

Článok (Bryson et al., 2003) opisuje DAML-S procesnú ontológiu, ktorá je okrem iného určená pre podporu návrhu zložených služieb. Opisuje aj rozšírenia DAML-S ontológie potrebné pre agentov.

Údaje

Aj keď údaje nie sú súčasťou DAML-S špecifikácie, sú kľúčové pre modulárny a agentovo orientovaný návrh. Niektoré údaje sú dôležitou súčasťou agenta a musia byť preto uložené v agentovi. Napríklad história agentových rozhodnutí, údaje o jeho postupe pri hľadaní.

Jedným z riešení by mohlo byť ukladať údaje do modulu podobného webovej službe, ktorý by bol prístupný iba pre agenta, pričom by sa zachovala štruktúra a rozhrania DAML a tým by sa zachovala aj jednotnosť kódovania.

Primitíva

Primitíva sú jednotlivé akcie na najjemnejšej úrovni podrobnosti, ktoré vykonáva webová služba. Sú dva základné spôsoby, ako sa primitíva môžu správať v systémoch reálneho času. V prvom prípade primitívum vykoná výpočet (ktorý môže trvať veľmi dlho) a vráti výslednú hodnotu. V druhom prípade primitívum naštartuje proces a vráti iba informáciu o tom, či sa podarilo úspešne naštartovať proces, alebo nie, kontrola stavu procesu a jeho výslednej hodnoty sú ďalšie akcie volajúceho programu.

BOD odporúča prienik medzi týmito dvoma prístupmi, ale aj keď DAML-S procesná ontológia umožňuje paralelné operácie, ich opis nie je dostatočný. Treba, aby sa pre každú možnú komunikáciu so službou špecifikovalo v profile služby, či dané volanie bude blokujúce alebo nie, či daný proces skončí za pevný, variabilný čas, alebo neskončí vôbec.

Postupnosti

Postupnosti sú jednoduchým a často používaným spôsobom (Laird & Rosenbloom, 1996) ako spájať elementy. V rámci BOD obsahuje POSH výber akcií dva druhy výberu akcií: vykonávané postupnosti, ktoré veľmi rýchlo vykonajú všetky svoje elementy v rámci jedného cyklu a akčné vzory, ktoré umožňujú kontrolu kontextu a zmenu priorít medzi jednotlivými elementmi. Ak jeden z elementov zlyhá, oba druhy postupností sú prerušené.

DAML-S obsahuje postupnosti, ale neurčuje, či sa dajú postupnosti prerušovať. Taktiež umožňuje, aby elementy postupnosti mali podprocesy, takže sa pomocou DAML-S postupností dajú reprezentovať iba akčné vzory a nie rýchlo vykonávané postupnosti.

Základné reaktívne plány

V dynamických prostrediach je výber akcií veľakrát príliš zložitý, aby sa dal reprezentovať pomocou postupností. V tých prípadoch sa dajú použiť základné reaktívne plány (BRP), ktoré majú väčšie vyjadrovacie možnosti.

DAML-S v súčasnosti neposkytuje možnosť priamo vyjadriť BRP, dajú sa vyjadriť iba nepriamo pomocou While konštrukcie a vnorených If-Then vetvení. Vzhľadom na výhodnosť BRP by bolo dobré, keby sa dali priamo vyjadriť v DAML-S.

1.2 Návrh scenárov pre sociálne agenty

Výsledkom práce pospolitosti vývojárov zaoberajúcich sa problematikou agentov a multiagentových systémov je niekoľko druhov agentov. Napríklad personálne agenty patria ľuďom a pomáhajú im pri obsluhe zložitých počítačových/ komunikačných systémov. V problematike multiagentových systémov sa zase stretávame s počítačovými agentami, ktorých cieľom je vzájomná interakcia. No problematike interakcie medzi ľuďmi a počítačovými agentami sa zatiaľ nevenovala dostatočná pozornosť. V tejto práci sa budeme zaoberať tzv. sociálnymi agentami, ktorých môžeme považovať za členov „ľudskej“ pospolitosti: sociálne agenty podporujú interakciu medzi ľuďmi, zatiaľ čo počítačové agenty podporujú interakciu ľudí a počítačov.

Príkladom môže byť konkrétny agent vyvinutý japonskou spoločnosťou, ktorý má podobu ženy a jeho cieľom je pracovať v TV, filmoch prípadne na internete. Na rozdiel od iného softvéru jeho hodnota nespočíva v jeho rýchlosti, správnosti, či efektívnosti, ale v tom, že reflektuje ľudské očakávania a záujmy. Takéto agenty nájdeme v rôznych online pospolitostiach alebo digitálnych mestách.

Aj keď je sociálny agent iba softvér, ľudia akceptujú jeho sociálnu rolu a tak sa môže správať ako sociálna bytosť vo virtuálnom svete. Veľká zmena nastane, keď ľudia vymenia svoje osobné agenty za sociálne. Od tohto okamihu nebude ich asistent (agent) výlučne len ich. Ak ich uznajú aj ostatní, používateľ bude nútený jednať so svojim agentom pomocou sociálnych protokolov. Keď sa sociálne agenty stanú súčasťou ľudských pospolitostí, nebude možné ignorovať to, čo hovoria alebo robia. Môžu poskytnúť svoje služby napríklad v oblasti bezpečnosti alebo navigovať ľudí do bezpečia v prípade katastrofy. Súčasne môžu negatívne vplývať na ľudí porušovaním sociálnych pravidiel, napríklad sledovaním, podvádzaním, či klamaním.

Aby sme pochopili problematiku sociálnych agentov, treba preskúmať platformu, ktorá nám umožní vykonať sériu experimentov s nimi. Nasledujúca časť obsahuje opis jazyka Q, ktorý slúži na opis scenárov správania agentov.

1.2.1 Jazyk Q

Q je jazyk určený na opis interakcií medzi agentami a používateľmi, ktoré sú založené na externých rolách agenta. Q je nezávislý od interných stavov agenta. Jeho cieľom je umožniť tvorcovi scenárov definovať požiadavky na správanie agenta.

Scenáre predstavujú most medzi návrhármi agentov (počítačoví profesionáli) a návrhármi aplikácií (autori scenárov). Na zvýšenie efektívnosti vytvorili kartu interakčných šablón (Interaction Pattern Card = IPC) založenú na rozhraní excelových tabuliek, ktorej cieľom je podporiť proces komunikácie medzi participujúcimi stranami pri vývoji formalizácie šablón správania agenta.

Jazyk Q je rozšírením jazyka Scheme, ktorý jedným z dialektov jazyka Lisp.

Podnety a akcie

Udalosť, ktorá vyvoláva interakciu agenta sa nazýva podnet (cue). Akcia (action) predstavuje požiadavku na vykonanie činnosti (zmenu prostredia) agenta.

Podnety a akcie sa v jazyku Q definujú takto:

```
(defcue name {(parameter in|out|inout)}*)
(defaction name {(parameter in|out|inout)}*)
```

Na rozdiel od programovacích jazykov, význam podnetov a akcií nie je v Q scenároch definovaný. Keďže podnety a akcie vykonávajú samotné agenty, ich význam úplne závisí od nich.

Príklad podnetov (začínajú na ?) a akcií (začínajú na !) :

```
(?pocujes "Ahoj" :od Jaro)
(!kracaj :od autobusova_stanica :k vlakova_stanica)
(!povedz "Ahoj" :k Jaro)
(?vidis vlakova_stanica :smer juh)
```

V predchádzajúcom príklade agent čaká na Jara, kým mu povie „Ahoj“. Následne prejde z autobusovej stanice k vlakovej, povie Jarovi „Ahoj“ a pozrie, či vidí na juhu vlakovú stanicu. Takéto akcie sa nazývajú synchronizované, čiže umožňujú agentovi pokračovať vo vykonávaní nasledujúcej akcie až po ukončení predchádzajúcej. Existujú aj asynchrónne akcie, ktoré sa môžu vykonávať paralelne viaceré naraz (v Q sa označujú pomocou !!).

Podmienené vykonávanie

Q pozná vetvenie na základe výskytu určitého podnetu podobne ako iné programovacie jazyky. Podmienku označuje kľúčové slovo `guard`:

```
(guard
     ((?pocujes "Ahoj" :od Jaro)
       (!povedz "Ahoj" :k Jaro) ...)
     ((?vidis vlakova_stanica :smer juh)
```

```
      (!kracaj :od autobusova_stanica :k vlakova_stanica) ...)
    (otherwise
      (!povedz "Stale cakam" :k Tomas))
)
```

V príklade ak agent počuje „Ahoj“ od Jara, odpovie mu „Ahoj“. Ak uvidí na juhu vlakovú stanicu, prejde k nej. Ak nezíska ani jeden z týchto podnetov, povie Tomášovi „Stále čakám“.

Scenáre

Scenáre slúžia na opis prechodov agenta z jedného stavu do druhého. Jednotlivé scenáre a môžu byť volané z iných scenárov či funkcií.

Ukážka definovania scenára:

```
(defscenario recepcia (sprava)
    (scene1 ((?pocujes "Ahoj" :od $x)
             (!povedz "Ahoj" :k $x) (go scene2))
            ((?pocujes "Dovidenia")
             (go scene3)))
    (scene2 ((?pocujes "Ahoj" :od $x)
             (!povedz "Ako vam pomozem?" :k $x))
            (otherwise (go scene3)))
    (scene3 ....)
)
```

V tomto scenári sa definuje niekoľko stavov (scene1, scene2, ...). Rovnaké podnety vedú k iným výsledkom pre rôzne stavy (napríklad na pozdrav reaguje agent inak v stave scene1 a scene2)

Agenty

Jazyk Q pozná niekoľko druhov agentov:

- Agent – predstavuje agenta realizujúceho definovaný scenár

```
(defagent hosteska :scenario 'recepcia)
```

- Avatar – ide o agenta riadeného priamo človekom

```
(defavatar Jaro)
```

- Dav (Crowd) – ide o dav viacerých agentov realizujúcich rovnaký scenár

```
(defcrowd chodec :scenario 'prechadzka :population 30)
```

1.2.2 Ukážky agentov

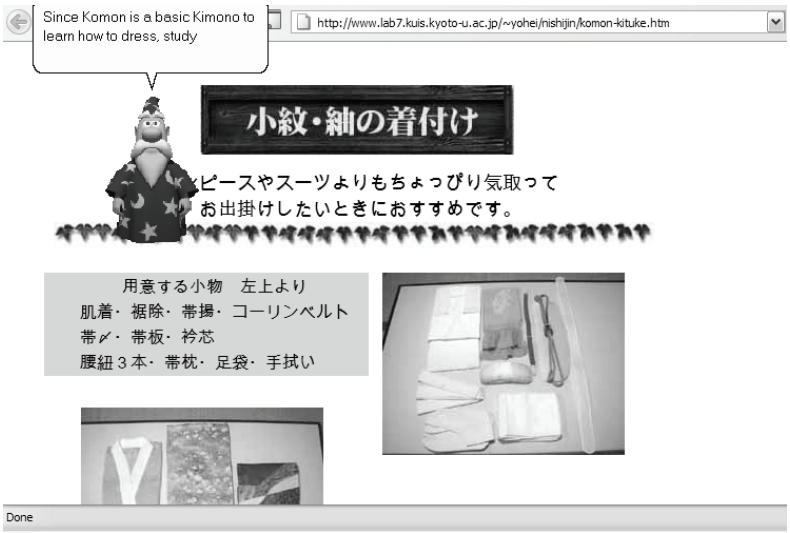
Nasledujúca časť obsahuje ukážky existujúcich sociálnych agentov v dvoch rôznych prostrediach: ako pomocníkov známych z operačných systémov Microsoft Windows a ako postavy vo virtuálnom 3D svete.

Microsoft agent

Na ilustráciu Q scenárov môžeme použiť relatívne známe agenty z prostredia Microsoft Windows. Ako príklad uvedieme pomocníka Merlina, ktorý bude realizovať tento scenár: Po príchode na stránku http://www.fiit.stuba.sk/ pozdraví používateľa gestom a opýta sa ho, čo by si chcel na stránke pozrieť. Q opis tohto scenára môže vyzerať nasledovne:

```
(scene1
     ((?see :address "http://www.fiit.stuba.sk/")
      (!behave :animation "Greet")
      (!speak "Ahoj, ja som Merlin.")
      (!fly :x 50 :y 300)
      (!speak "Vitajte na strankach Fakulty informatiky a
               informacnych technologii")
      (!speak "Co by ste si chceli na stranke pozriet")
      (!ask :selection '("Novinky", "Informacie pre uchadzacov",
                         "Informacie pre studentov"))
      (!receive $x)
      (go scene2)))
     ...
```

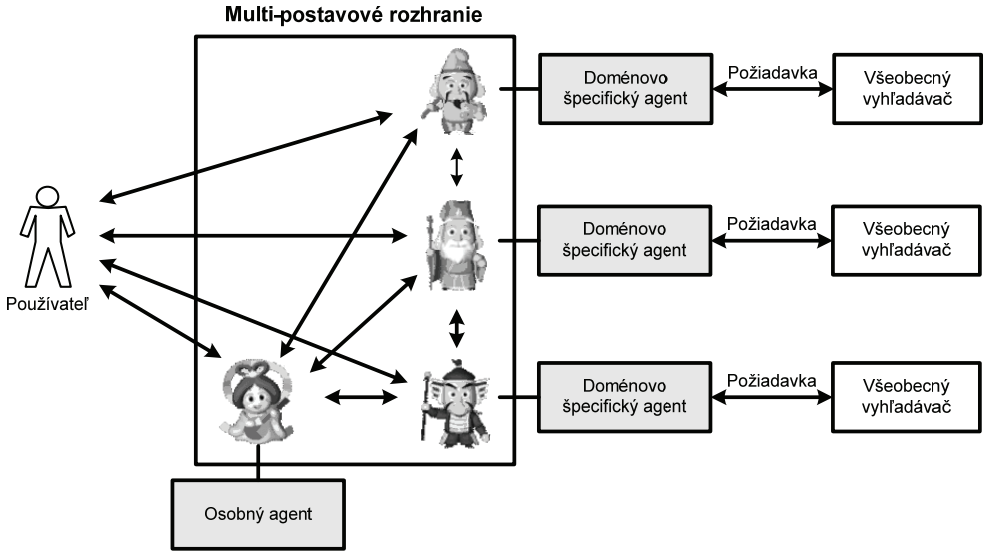
Ukážka realizácie podobného scenára sa nachádza na obrázku 1-1. V tomto prípade informuje Merlin o odeve Kimono.



Obrázok 1-1. Príklad MS Agenta Merlin pri prezentovaní informácií o Kimone (Zhong et al., 2003).

S použitím jazyka Q a agentov od Microsoftu bolo vytvorené multi-postavové rozhranie na získavanie informácií, v ktorom doménovo špecifické vyhľadávacie agenty spolupracujú pri plnení používateľských požiadaviek. Predchádzajúce riešenia podobného problému využívali na svoju činnosť tabuľovú architektúru, ktorá umožňovala integrovanie výsledkov práce viacerých agentov. Pri tomto riešení bol použitý iný prístup: namiesto ukladania informácií do spoločného úložiska sa integračný proces zobrazí používateľovi vo forme dialógu viacerých postáv

reprezentujúcich jednotlivé agenty. Používateľ môže sledovať ich spoluprácu a v prípade potreby priamo zasiahnuť do ich dialógu. Multi-postavové rozhranie zvyšuje spokojnosť používateľa tým, že integruje spoluprácu jednotlivých agentov pred jeho očami sociálnym spôsobom. Na obrázku 1-2 sa nachádza ukážková schéma spolupráce viacerých agentov.



Obrázok 1-2. Schéma spolupráce viacerých agentov.

FreeWalk agenty

Ďalšou ukážkou použitia sociálnych agentov je prostredie FreeWalk. Ide o 3D webový simulátor virtuálnych miest s virtuálnymi obyvateľmi, v rámci ktorého sa dajú vytvárať virtuálne kópie existujúcich miest. Ukážka z tohto prostredia sa nachádza na obrázku 1-3.

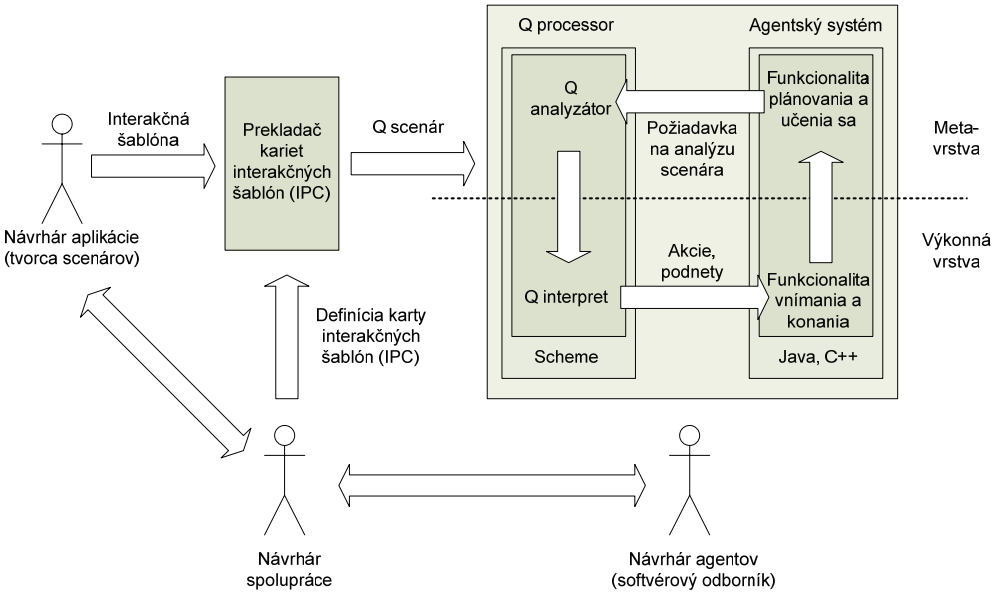
Prostredie umožňuje vytvárať veľké množstvo agentov, ktoré sa pohybujú v reálnom čase v jednotlivých mestách. S ich použitím a pridaním niekoľkých agentov ovládaných priamo používateľmi môžeme robiť experimenty so simuláciou rôznych situácii, ktoré by sa v reálnom svete len veľmi ťažko uskutočňovali.

Obrázok 1-3. Ukážka prostredia FreeWalk spolu s agentami (Zhong et al., 2003).

1.2.3 Vývoj scenárov

Q architektúra

Obrázok 1-4 zachytáva architektúru spracovávania scenárov. Keď sa scenár pridelí konkrétnemu agentovi, vytvorí sa Q – procesor zviazaný so zodpovedajúcim agentským systémom. Agentský systém dokáže byť hostiteľom viacerých agentov (napríklad spomínaný FreeWalk dokáže byť hostiteľom stoviek agentov). Tu môže vznikať problém pri paralelnom (súbežnom) vykonávaní viacerých agentov, v rámci ktorého môžu viesť viaceré behy rovnakých scenárov k rôznym výsledkom. Aby sme sa vyhli tomuto nedostatku, použijeme nástroje jazyka Scheme (materského jazyka Q), ktoré umožňujú riadiť prepínanie medzi jednotlivými konkurenčnými procesmi.



Obrázok 1-4. Q architektúra.

Aj keď je jazyk Q implementovaný ako nadstavba jazyka Scheme, existujú rozhrania pre C++ a Javu umožňujúce kombinovať Q a iné agentské systémy.

Architektúra Q je rozdelená do dvoch vrstiev: *vykonávacej vrstvy (execution layer)* a *meta-vrstvy (meta-layer).* Vo vykonávacej vrstve Q interpret vyhodnocuje podnety a akcie podľa definovaného scenára. Ak pri tom narazí na problém v scenári, prepne sa do meta-vrstvy, v ktorej Q analyzátor vyšetrí scenár. Na základe výsledkov Q analyzátora môže dôjsť ku komunikácii s tvorcami scenárov.

Agenty môžu byť autonómne alebo závislé. V prípade autonómnych sú scenáre jednoduché, v prípade závislých špecifikujú všetky detaily. Všimnime si, že všeobecnosť (granularita) podnetov a akcií závisí od dvoch nezávislých faktorov: miery autonómie agenta a stupňa presnosti, ktorý potrebujú tvorcovia scenárov.

Proces návrhu

Jazyk Q predstavuje rozhranie medzi tvorcami scenárov (návrhári aplikácie) a softvérovými profesionálmi (návrhári agentov), ktorí predstavujú dve úplne odlišné perspektívy.

Proces vytvárania scenárov pozostáva z týchto krokov:

1. Tvorcovia scenárov a agentov sa dohodnú na podnetoch a akciách, čím vznikne „slovná zásoba“ pre tvorcov scenárov.
2. Autori scenárov opíšu jednotlivé scenáre pomocou jazyka Q, zatiaľ čo autori agentov implementujú podnety a akcie.
3. Môžeme zadefinovať aj tretiu rolu v rámci procesu vývoja: návrhára spolupráce, ktorého úlohou je sledovať vzory spolupráce v jednotlivých doménach a vytvárať karty IPC (Interaction Pattern Card). Vývoj kariet spúšťa dialóg medzi návrhárom spolupráce a autormi scenárov, čo vedie k lepšiemu pochopeniu vzorov spolupráce v jednotlivých doménach. Rovnako IPC karty zvyšujú produktivitu tvorcov scenárov.

1.2.4 Použitie sociálnych agentov

Simulácia krízového manažmentu

Jednou z konkrétnych aplikácií sociálnych agentov je ich využitie pri simulovaní krízových situácií. Agenty v týchto simuláciách zastávajú role osôb bežného života ako chodcov, bezpečnostných zložiek a podobne. Realistická simulácia evakuácie sa dosiahne, keď necháme pohybovať sa chodcov v prostredí snažiac sa uniknúť. Takéto simulácie odhalia typické chyby v správaní sa ľudí v daných situáciách a pomôžu nájsť ich najlepšie zvládnutie. Príklad konkrétneho scenára môže vyzerať takto:

Evakuácia železničnej stanice a stanice metra vo virtuálnom Kyote

1. Rok 200X. Simulácia evakuácie za pomoci stoviek sociálnych agentov a približne 20 používateľov ovládajúcich agentov. Údaje získané zo simulácie sa uložia a vyhodnotia.
2. Rok 200Y. Stane sa katastrofa a na stanici vypukne požiar. Situáciu zaznamenaná riadiace stredisko. To na základe predošlých simulácií odošle príkazy mobilným terminálom ľudí a agenty ich následne začnú navigovať do bezpečia.

Aby tieto experimenty mali zmysel, treba skúmať podobnosť správania sa agentov a skutočných ľudí. Príkladom môže byť vytvorenie simulácií menších rozmerov vo fyzickom aj virtuálnom svete a ich vzájomné porovnanie a minimalizovanie rozdielov. Taktiež treba skúmať rozdiely v spolupráci človek-človek a človek-agent. Softvéru veríme lebo je efektívny, no ľuďom veríme nie preto, že sú efektívni. Preto sa môžu ľudské reakcie líšiť, keď im tú istú informáciu podá človek alebo agent.

Sociálno-psychologický výskum agentov

Aby sme pochopili podstatu agentov, vykonali sa viaceré sociálno-psychologické experimenty. Momentálne sa zameriavajú na zistenie, do akej miery môžu agenty podporiť medziľudskú komunikáciu a ovplyvniť medziľudské vzťahy. Agenty môžu pôsobiť ako prostredníky medzi ľuďmi v rôznych sociálnych rolách, napríklad medzi obyvateľmi a návštevníkmi, staršími a mladšími a podobne.

Vykonali sa konkrétne experimenty, na ktorých sa zúčastnili americkí a japonskí študenti, ktorí s pomocou agentov medzi sebou diskutovali.

1.2.5 Zhodnotenie

Vývoj jazykov opisujúcich správanie sa agentov napreduje. Nedošlo k ich výraznejšiemu použitiu v aplikáciách. Preto vytvorili jazyk Q opisujúci interakcie medzi človekom a agentom. Ako ukážky použiteľnosti jazyka Q vytvorili jeho aplikácie pre prostredie Microsoft Agents a FreeWalk. Aj keď ide o úplne odlišné druhy agentov, mohli sa riadiť pomocou jazyka Q, pretože ten umožňuje definovanie požiadaviek pre nezávislé autonómne agenty.

Rozdiely medzi počítačovými a sociálnymi agentami sú zobrazené v tabuľke 1-3. Na rozdiel od počítačových agentov, sociálne nepredpokladajú správnosť multi-agentových scenárov. Radšej kombinujú vykonávaciu a meta-vrstvu, čo vytvára robustné správanie agentov. Chyby v scenároch sa neopravujú ladením ale skúšaním.

Tabuľka 1-3. Porovnanie počítačových a sociálnych agentov.

	Počítačové agenty	Sociálne agenty
Model	Interný mechanizmus	Vzor správania
Opis	Program	Scenár
Tvorca	Počítačový odborník	Tvorca scenárov
Cieľ	Správnosť	Robustnosť
Testovanie	Debugovanie zdrojového kódu	Skúšanie
Výsledok	Počítačové spoločenstvo	Agentami sprostredkovaná komunita

S narastajúcou koexistenciou ľudí a agentov na internete vzniká potreba definovania multiagentových scenárov určených na realizáciu rôznych druhov webových aplikácií. Ak by sme dovolili agentom úplne nezávislé správanie, celý systém by sa ťažko riadil. Preto je potrebné špecifikovať sociálne ohraničenia ako scenáre, ktoré riadia ich správanie.

2 DOLOVANIE V DÁTACH NA WEBE

Webový priestor je bohatým zdrojom veľkého množstva dát a informácií, ktoré sú vzájomne prepojené. Použitím rôznych prístupov sú v tejto časti opísané rôzne techniky dolovania v dátach na webe. Ich spoločným cieľom je získavanie nových informácií a znalostí z webových dát.

V úvode je opísaná technika vyhľadávania informácií porovnávaním webových stránok. Zisťovaním nepriamych asociácií z dát o používaní webu sa zaoberáme v druhej časti. Opisujeme nové techniky dolovania v dátach nazývané nepriame asociácie. Problematika obalovačov, ich kategorizácia a spôsob ich vytvorenia je uvedený v tretej časti. Nakoniec sa venujeme dolovaniu webových záznamov, kde je uvedený stručný prehľad dolovania v dátach, popis predprípravy dát a analýza vzorov.

2.1 Vyhľadávanie informácií porovnávaním webových stránok

Táto časť sa zoberá problematikou vyhľadávania informácií na webových stránkach. Pojednáva o najpoužívanejšom prístupe založenom na vyhľadávaní pomocou kľúčových slov. Tento prístup možno označiť istým spôsobom ako nedostatočný a z toho dôvodu sa navrhuje iný vhodnejší spôsob vyhľadávania.

Navrhovaný prístup vyhľadávania nie je možné vo všetkých prípadoch považovať za dostatočný vo všetkých smeroch, nakoľko sa zameriava na vyhľadávanie iného typu informácií tzv. „neočakávaných informácií“, pričom v súčasnosti bežnými spôsobmi vyhľadávania sa dá k takýmto informáciám len zdĺhavým postupným prehľadávaním webových stránok.

2.1.1 Problematika vyhľadávania na webe

Idea vyhľadávania informácií na webe nie je nová. Od začiatku vzniku internetu sa vyhľadávanie informácií ukazuje ako jedna z najdôležitejších požiadaviek.

V súčasnosti je najviac rozšírené vyhľadávanie založené na vyhľadávaní pomocou kľúčových slov, prípadne menej rozšírený spôsob – vyhľadávanie „šité na mieru“ (tzv. pomocou *obaľovačov*). Systémy, ktoré využívajú vyhľadávanie pomocou kľúčových slov umožňujú používateľovi vyhľadať informáciu špecifikovanú množinou kľúčových slov, pričom potom veľmi zjednodušene povedané vyhľadávací systém vyhľadá všetky webové stránky, ktoré obsahujú zadané kľúčové slová.

Základným obmedzením týchto prístupov je vlastnosť, ktorá vychádza z podstaty vyhľadávania pomocou kľúčových slov a to špecifikácia vyhľadávanej informácie.

Nakoľko vyhľadávanú informáciu používateľ špecifikuje dopredu, dá sa povedať, že vie dopredu akú informáciu požaduje a akú informáciu mu má vyhľadávací systém nájsť. Takáto informácia sa môže označiť termínom očakávaná informácia. Namiesto očakávaných informácií by v mnohých prípadoch mohlo byť výrazne prospešnejšie, pokiaľ by bol systém schopný nájsť informácie, ktoré nie sú priamo špecifikované (z dôvodu nedostatočných vedomostí používateľa) a teda sú v istom zmysle neočakávané.

Ako sme už načrtli, vyhľadanie takýchto informácií bežnými spôsobmi by zahŕňalo prehľadanie množstva webových stránok. Použitie špeciálnych systémov na vyhľadanie takýchto informácií by mohla ovplyvniť najmä špecifickosť, alebo subjektívne vnímanie a rozlišovanie neočakávanej informácie.

2.1.2 Vlastnosti informácií

Aby sme mohli ďalej ozrejmiť navrhnutý spôsob vyhľadávaných informácií, treba zadefinovať určité vlastnosti vyhľadávaných informácií.

- Neočakávanosť informácie. Vlastnosť informácie vyjadrujúca, že daná informácia je relevantná, ale pre používateľa istým spôsobom neznáma.
- Použiteľnosť informácie. Časť informácie je použiteľná používateľom (ak ju použije. môže to pre neho znamenať určitú výhodu).

Napriek tomu, že neočakávanosť je dôležitá vlastnosť, použiteľnosť je vlastnosť, ktorú musia spĺňať všetky informácie nájdené pomocou akéhokoľvek vyhľadávača (na základe tohto kritéria sa dá hodnotiť úspešnosť práce vyhľadávača).

Na základe uvedených tvrdení môžeme vyhľadávané informácie klasifikovať takto:

- Informácie, ktoré sú očakávateľné a zároveň použiteľné (z hľadiska opisovanej idey vyhľadávania je tento typ informácií najdôležitejší).
- Informácie, ktoré sú neočakávateľné a zároveň nepoužiteľné (sú to napríklad informácie, ktoré nesúvisia s vyhľadávanou témou).
- Informácie, ktoré sú síce použiteľné ale zároveň očakávateľné.

2.1.3 Idea navrhnutého riešenia

Informačný priestor môžeme rozdeliť do troch celkov:

- Stránky používateľa U reprezentujúce jeho súčasné znalosti. Môžu to napríklad byť odkazy na webové stránky, ktoré sa zaoberajú hľadanou problematikou a obsahujú informácie o ktorých používateľ vie, hodnotí ich ako relevantné k téme a v istom zmysle reprezentujú jeho rozsah znalostí.
- Kľúčové slová E predstavujú dodatočnú špecifikáciu vyhľadávaných informácií.
- Prehľadávaný informačný priestor C.

Potom sa vyhľadáva takto:

1. Analýza používateľom zadaných znalostí a ich vnútorná reprezentácia v systéme.
2. Analýza prehľadávaného priestoru C vzhľadom na U a E a na základe toho vrátenie správnych výsledkov vyhľadávania.

V tomto bode sa ale vynára otázka vnútornej reprezentácie extrahovaných znalostí. Vzhľadom na to, že systém na vyhľadávanie je určený pre relatívne malé informačné priestory, zvolili sa dva spôsoby reprezentácie dokumentov:

- vektorová reprezentácia, kde sa dokument reprezentuje vektorom kľúčových slov,
- asociačnými pravidlami, kde sa dokument reprezentuje vzťahmi jednotlivých kľúčových slov.

Vektorová reprezentácia dokumentov

Vektorová reprezentácia textových dokumentov sa bežne používa v oblasti získavania a vyhľadávania informácií. Jej základom je reprezentácia ľubovoľného textového dokumentu množinou kľúčových slov pomocou vektoru (Poznámka: Jednotlivé kľúčové slová sa reprezentujú základným gramatickým tvarom.). Vektor $K=k_1, k_2,..., k_t$, kde k_i je jedno konkrétne kľúčové slovo, reprezentuje dokument s *t* kľúčovými slovami. Ku každému kľúčovému slovu k_i z dokumentu d_j sa priradí váha $w_{ij} > 0$, pokiaľ sa kľúčové slovo nachádza v dokumente, inak $w_{ij} = 0$. Potom je možné ľubovoľný dokument d_j reprezentovať váhami kľúčových slov

$$d_j = w_{1,j}, w_{2,j},...,w_{t,j} .$$

Potom je možné definovať operáciu podobnosti

$$sim(d_j,q) = (d_j * q) / (|d_j| \times |q|),$$

kde q je vektor reprezentujúci vyhľadávaciu požiadavku.

Možnosť, ako sa dá ovplyvniť reprezentácia dokumentu a tým aj výsledky vyhľadávania je ovplyvnenie váhy jednotlivých kľúčových slov rôznymi spôsobmi ich výpočtu:

- TF-IDF váhová schéma

  $W_{i,j} = tf_{i,j} \times \log (N/n_i)$,

  kde

  N – celkový počet dokumentov

  n_i – počet dokumentov, v ktorých sa k_i vyskytuje

  f_i – frekvencia výskytu k_i

  $tf_{i,j} = f_{i,j} / (\max_i f_{i,j})$ je normalizovaná frekvencia pre dokument d_j

- Schéma podľa Saltona a Buckleyho

  $W_{i,j}=(0,5 + (0,5 \times f_{i,q})/\max(If_{i,q})) \times \log(N/n_i)$,

  kde

  $Idf_i = \log (N/n_i)$ je inverzná frekvencia dokumentu pre kľúčové slovo k_i

Reprezentácia dokumentu asociačnými pravidlami

Asociačné pravidlá predstavujú významný model v technikách dolovania v dátach. Typickým príkladom je analýza nákupného košíka. Zahŕňa analýzu položiek nákupného košíka a vzťahov medzi nimi. Typickým príkladom je asociačné pravidlo

$$syr \rightarrow pivo \text{ [podpora = 10\%, významnosť = 80\%]}.$$

Tento príklad asociačného pravidla vyjadruje, že 10% zákazníkov zakúpi pivo a syr naraz a tí ktorí kupujú syr na 80% kúpia aj pivo.

Model asociačných pravidiel je definovaný nasledovným spôsobom. Nech I = i_1, i_2,...,i_m je množina položiek. Nech T je množina transakcií, kde každá transakcia je množina t a tá je podmnožinou I. Asociačné pravidlo je implikácia X→Y, kde X je podmnožinou I a súčasne Y je podmnožinou I pričom musí platiť X ∩ Y = 0. Asociačné pravidlo X→Y sa vyskytuje s významnosťou c% z transakcií T. Pravidlo má podporu s% v T, pokiaľ s% z transakcií T obsahuje X U Y.

Problém v dolovaní v dátach pomocou asociačných pravidiel je rozpoznanie pravidiel, ktoré prekračujú minimálne hodnoty podpory a významnosti.

Následne použitie opísanej metódy pozostáva z dvoch krokov:

1. vygenerovanie množiny z kľúčových slov,
2. vygenerovanie asociačných pravidiel spĺňajúcich minimálne kritériá.

V navrhnutom riešení sa môžu vygenerovať asociačné pravidlá z každej webovej stránky z C a z každej webovej stránky z U samostatne. Dôvodom takéhoto prístupu je možnosť pozerať na rôzne stránky z rôznych pohľadov (čo sa týka informácií). Pokiaľ by došlo k pomiešaniu U a C mohlo by dôjsť k strate informácií.

2.1.4 Navrhnuté techniky vyhľadávania

Vyhľadávanie podobných webových stránok

Vyhľadávanie podobných webových stránok sa zakladá na vyhľadávaní podobných stránok z C k stránkam U. Tento prístup je vhodný, pokiaľ používateľ chce vykonať detailnú analýzu špecifikovanej témy.

Porovnanie sa zakladá na kosínusovom porovnaní podobnosti danej stránky u_j z U s každou stránkou z C. Nasledujúci príklad demonštruje vyhľadanie pomocou tejto opisovanej techniky.

Stránky U

- Stránka U1: (dáta, 1), (predpoklad, 1)
- Stránka U2: (informácia, 2), (extrahovanie, 2), (dáta, 2)
- Stránka U3: (klasifikácia, 2), (pravdepodobnosť, 2)
- Stránka U4: (zhlukovanie, 2), (segment, 1)

Stránky C

- Stránka C1: (dáta, 2), (predpoklad, 2), (klasifikácia, 3)
- Stránka C2: (asociácia, 3), (dolovanie, 2), (pravidlo, 1)
- Stránka C3: (zhlukovanie, 3), (segment, 2), (dáta, 2)

Výsledkom vyhľadávania v stránkach C vzhľadom na stránku U1 sú stránky s nasledujúcim poradím hodnotenia:

- Stránka C1
- Stránka C3

Vyhľadanie neočakávaných výrazov z C s ohľadom na U

V mnohých prípadoch chce používateľ poznať neočakávané výrazy zadaných dvoch podobných stránok, jedna z C a podobných z U. Tieto výrazy umožňujú používateľovi zistiť kľúčové rozdiely dvoch stránok., pričom tieto môžu následne napomôcť používateľovi v rozhodovaní, či bude pokračovať v presmerovaní na konkrétnu stránku.

Spôsob porovnania sa môže zakladať na porovnaní stránky u_j z U a stránky c_i z C porovnaním váh oboch dokumentov vzhľadom na výrazy u_j.

Pokiaľ by sa pokračovalo v predchádzajúcom príklade a porovnávala by sa stránka C1 vzhľadom na stránku U1, zistilo by sa, že stránky sú podobné a neočakávaný výraz by bol „*klasifikácia*“ (nakoľko „*dáta*“ a „*predpoklad*“ sa vyskytujú v U1), pričom pokiaľ by bolo viacero výsledkov, usporiadali by sa podľa hodnotenia.

Vyhľadanie neočakávaných stránok z C s ohľadom na U

Základom tejto metódy je vyhľadanie množiny neočakávaných webových stránok z množiny C s ohľadom na znalosti reprezentované množinou webových stránok U. Mnoho krát môžu byť takéto stránky veľmi zaujímavé, nakoľko používateľ vôbec o nich a o informáciách v nich obsiahnutých nemusí vôbec vedieť (keďže ich nedefinoval ako svoje znalosti).

Algoritmus vyhľadávania neočakávaných stránok z C s ohľadom na U:

1. Kombinácia všetkých webových stránok z množiny U do tvaru tvoriaceho jeden dokument Du
2. Pre každú webovú stránku c_i z množiny C sa vypočíta hodnota reprezentujúca neočakávanosť stránky c_i pomocou vzorca

$$unexpC_i = (\sum_{r=i}^{m} unexpT_{cki}) / m$$

kde webová stránka $c_i = \{k_1, k_2, ..., k_m\}$ sa reprezentuje kľúčovými slovami a $unexpT_{cki}$ je početnosť neočakávaného kľúčového slova, pričom ak je kľúčové slovo očakávané $unexpT_{cki} = 0$.

Vzhľadom na predchádzajúci príklad by výsledok vyhľadávania v množine C bol usporiadaný takto:

1. Stránka C2
2. Stránka C3
3. Stránka C1

Vyhľadanie stránok s neočakávaným obsahom z C vzhľadom na U

V mnohých prípadoch výskyt určitých kľúčových slov na webovej stránke nemusí zaručovať, že daná stránka sa zaoberá určitou problematikou. Zvýšenie pravdepodobnosti vyhľadania v tomto smere sa môže dosiahnuť analýzou výskytu kľúčových slov v jednej vete, pričom nie je dôležité, v akom poradí sa tieto slová v danej vete vyskytujú (napríklad: „dáta sú extrahované“ alebo „extrakcia dát“ reprezentujú výskyt dvoch kľúčových slov extrakcia a dáta).

Na tento účel sa použil algoritmus extrakcie dát pomocou asociačných pravidiel.

Vyhľadanie neočakávaných odkazov z C s ohľadom na U

Tento prístup slúži na vyhľadávanie všetkých odkazov zo stránky z množiny C, ktoré sa nevyskytujú na stránkach množiny U. Tento prístup nie je ekvivalentný predchádzajúcim spôsobom vyhľadávania, nakoľko má slúžiť ako pomôcka pri prehliadaní určitej webovej stránky a smerovaní ďalšieho manuálneho vyhľadávania.

2.1.5 Vyhodnotenie

V tejto časti sme predstavili rôzne prístupy k vyhľadávaniu informácií v priestoroch webových stránok. Z opísaných metód a použitých algoritmov vyplýva, že tieto prístupy sú skôr vhodné pre menšie webové priestory (rádovo stovky – tisíce stránok). Dôvodom je vlastnosť opísaných algoritmov – reprezentácia jednotlivých dokumentov pomocou kľúčových slov, alebo uchovávanie jednotlivých asociačných pravidiel pre každý dokument. Dôsledkom uvedeného môže byť spôsob nasadenia takéhoto systému v závislosti od požiadaviek používateľa na vyhľadávací systém.

2.2 Zisťovanie nepriamych asociácií z dát o používaní webu

Web má veľký vplyv na všetky aspekty spoločnosti. Stal sa dôležitým zdrojom na získavanie informácií a hodnotným prostriedkom na ich zhromažďovanie. Sledovanie pohybu používateľov na webe poskytuje obrovské možnosti pre komerčné spoločnosti, aby získali a zhromažďovali informácie o priamych používateľoch webu. Pri zvyšovaní počtu používateľov vzrastá aj množstvo informácií zhromaždených webovými servermi. Zvýšil sa aj záujem o aplikovanie techník dolovania na objavovanie skrytých znalostí v dátach, v ktorých sú zaznamenané informácie o pohybe používateľov na webe.

Asociačné pravidlá a sekvenčné asociácie sú dva významné typy vzorov webu, ktoré môžu byť hodnotným prínosom pre komerčné organizácie. Tieto webové asociačné vzory môžu byť využité na nasledujúce účely:

- Na zhromažďovanie informácií o správaní sa používateľov webu na obchodné účely. Tieto informácie môžu použiť na efektívny rozvoj marketingových kampaní cielených na určité skupiny používateľov webu.
- Na predpoveď, aká bude nasledujúca stránka, ktorú používateľ navštívi.
- Na spoluprácu s administrátormi pri reorganizácii webu.

Predchádzajúce práce v tejto oblasti sa zamerali na hľadanie vzorov, ktoré sa často objavujú v dátach, t.j. dáta s dostatočne vysokou početnosťou. Rôzne vzory, ktoré nedosahujú minimálny prah početnosti, majú predpoklad byť štatisticky nevýznamné. Súčasne treba zaznamenať, že niektoré nie veľmi časté vzory môžu poskytnúť dodatočné porozumenie najmä v tých dátach, ktoré sa spájajú s negatívne asociovanými vzormi.

Asociačné pravidlá (Agrawal et al., 1993) a sekvenčné asociácie (Agrawal & Spikant, 1995) navrhli na analyzovanie transakcií zákazníkov tzv. nákupný košík, kde nie sú žiadne obmedzenia určujúce aký typ položky môže byť nakúpený spolu.

Napríklad webové stránky, ktoré sú umiestnené blízko domovskej stránky, sú navštevované častejšie v porovnaní so stránkami umiestnenými ďalej.

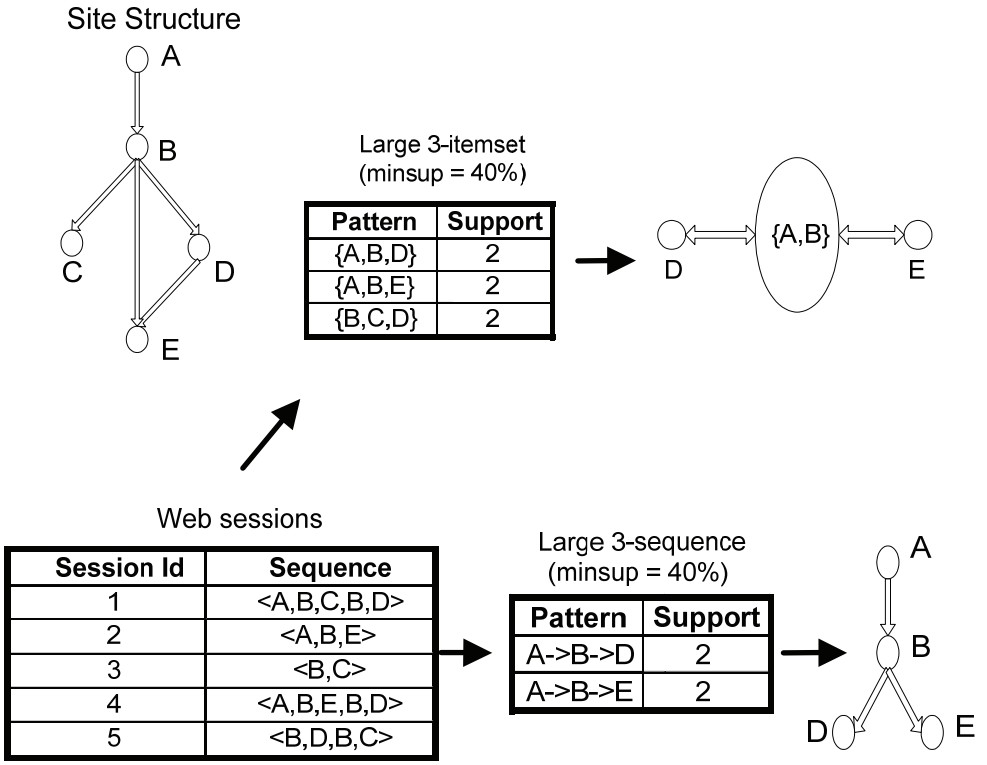
2.2.1 Nepriame asociácie

V tejto časti sa použijú nové techniky dolovania v dátach, nazvané nepriame asociácie, na dáta obsahujúce postupnosť klikov používateľa webu. Tieto techniky boli pôvodne vyvinuli v (Tan et al., 2000a) pre transakciu typu nákupný košík.

Základnou myšlienkou nepriamych asociácií je hľadanie dvojice binárnych premenných, ktoré sú navzájom negatívne asociované, ale obidve sú pozitívne asociované so spoločnou sadou položiek nazývaných mediátor – sprostredkovateľ. Tieto vzory sa môžu získať kombinovaním častých vzorov, ktoré majú podobnú subštruktúru do kompaktnejších vzorov.

Na ilustráciu tohto prístupu uvedieme príklad:

Dáta obsahujú päť unikátnych transakcií. Použitím štandardných asociačných pravidiel a sekvenčných asociačných algoritmov s využitím podpory 40% sa zistili časté položky a časté podúseky.



Obrázok 2-1. Príklad nesekvenčných a sekvenčných nepriamych asociácií (Zhong et al., 2003).

Pre nesekvenčné asociácie sme zistili, že stránka D sa objavuje často s oboma stránkami A a B, podobne stránka E.

Bez žiadnych predchádzajúcich znalostí o štruktúre a obsahu webových stránok môžeme predpokladať, že stránky D a E sú často navštevované, napriek tomu, že (B,E) je pod hranicou minima (v príklade má spojenie D a E podporu 1). Hovoríme, že stránky sú nepriamo asociované cez sprostredkovateľa {B,D}.

Pre sekvenčný prípad zistíme, {A}→{B}→{D} a {A}→{B}→{E} sú časté krížové spojenia, zatiaľ čo D a E sa spolu často neobjavujú, sú nepriamo asociované cez sekvenciu {A}→{B}.

Poznámka:

Každá sekvenčná nepriama asociácia má porovnateľnú nesekvenčnú nepriamu asociáciu, avšak k niektorým nesekvenčným nepriamym asociáciám neexistuje sekvenčná nepriama asociácia.

Negatívne asociácie medzi nepriamo asociovanými webovými stránkami sú zaujímavé, lebo očakávame, že ich podpora bude vysoká, čo je dané spoločným sprostredkovateľom.

2.2.2 Definície

Definícia: Nech $I=\{i_1,\ldots,i_d\}$ je neprázdna množina všetkých položiek (javov, udalostí) dostupných v databáze. Každá neprázdna podmnožina I sa nazýva položková množina. Ak položková množina obsahuje k položiek, nazývame ju *k* položkovou množinou.

Napríklad I={a,b,c,d,e}, potom C={b,c} je príkladom dvojpoložkovej množiny, ktorá je podmnožinou I.

Definícia: Nech T je množina všetkých transakcií, kde každá transakcia $t \in T$ je súborom položiek. Nech transakcia t obsahuje položkovú množinu C a $C \subseteq t$. V tomto prípade podpora C je definovaná ako časť transakcií, ktoré obsahujú C. Označujeme ju sup (C).

Definícia: Položkovú množinu nazývame veľkou, alebo frekventovanou, ak jej podpora je väčšia ako minimálna hranica t_f špecifikovaná používateľom.

Definícia: Sekvencia je usporiadaný zoznam položkových množín $s = s_1 \rightarrow s_2 \rightarrow \ldots \rightarrow s_n$, kde každú položkovú množinu s_j nazývame prvkom sekvencie.

Počet položiek, ktoré patria do s_j označujeme $|s_j|$. Ak celkový počet položiek, ktoré patria do sekvencie s sa rovná k, t.j. $k = \Sigma|s_j|$, potom s nazývame k sekvenciou. Dĺžku sekvencie označujeme $|s|$ a rovná sa počtu prvkov obsiahnutých v s.

Napríklad s = {a} → {a,c} → {b} obsahuje štyri položky a tri prvky. Preto je to štvorpoložková množina so sekvenčnou dĺžkou rovnou tri.

Sekvencia je neprázdna, ak obsahuje aspoň jeden prvok, teda $|s| > 0$.

Položka sa môže objaviť viackrát v rôznych prvkoch sekvencie, ale iba raz v rámci jedného prvku.

Definícia: Položku X, ktorá sa objaví presne jedenkrát v sekvencii, s nazývame unikátna, alebo neopakujúca sa položka.

Napríklad b,c sú neopakujúce sa položky v sekvencii s = {a} → {a,c} → {b}.

Definícia: Ak existuje prirodzené číslo $1 \le j_1 < j_2 < \ldots < j_n \le n$ a platí: $t_1 \subseteq s_{j1}, t_2 \subseteq s_{j2}, \ldots t_n \subseteq s_{jn}$, t nazývame podsekvenciou sekvencie s.

Napríklad t = {c} → {b } je podsekvenciou sekvencie s = {a} → {a,c} → {b}.

Definícia: Zreťazenie dvoch sekvencií s a t je sekvencia dĺžky $|s|+|t|$ a obsahuje všetky prvky s, za ktorými bezprostredne nasledujú prvky t, označujeme st.

Definícia: Sekvencia w je predponou sekvencie s, ak existuje neprázdna sekvencia y taká, že s = wy, označujeme w [s.

w_1 je minimálnou predponou sekvencie s, ak w_1 [s a $|w_1| = 1$.

w_2 je maximálnou predponou sekvencie s, ak w_2 [s a $|w_2| = |s| - 1$.

Napríklad s = {a} → {a,c} → {b} je sekvencia, t_1 = {a} → {a,c } je maximálna predpona sekvencie s, {a} je minimálna predpona sekvencie s. t_2 = {a} → {a }nie je predpona sekvencie s.

Definícia: sekvencia y je príponou sekvencie s, ak existuje neprázdna sekvencia w, taká, že s = wy, označujeme y] s.

y_1 je minimálnou príponou sekvencie s, ak y_1] s a $|y_1| = 1$.

y_2 je maximálnou príponou sekvencie s, ak y_2] s a $|y_2| = |s| - 1$.

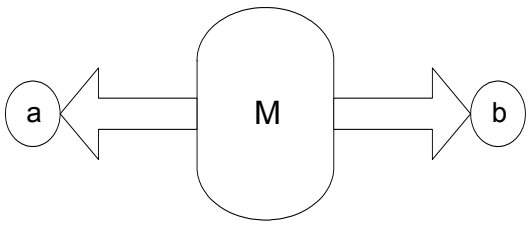
Napríklad t_3 = {b} je minimálnou príponou sekvencie s = {a} → {a,c} → {b},

t_4 = {a,c} → {b} je maximálnou príponou sekvencie s.

2.2.3 Nesekvenčné nepriame asociácie

Definícia: Webové stránky *a* a *b* sú nepriamo asociované cez množinu M (mediátor), ak sú splnené nasledujúce podmienky: Podpora ({a,b}) < t_s, Podpora ({a} ∪ M) ≥ t_f, Závislosť (a,M) ≥ t_d, Závislosť (b,M) ≥ t_d (Brijs et al., 1999, Brin et al., 1997a, Tan & Kumar, 2000b, Tan et al., 2002).

Závislosť (x,M) môže byť prijateľná objektívna jednotka, t_s, t_f, t_d sú určené hranice závislosti a podpory.



Obrázok 2-2. Nesekvenčné nepriame asociácie.

2.2.4 Sekvenčné nepriame asociácie

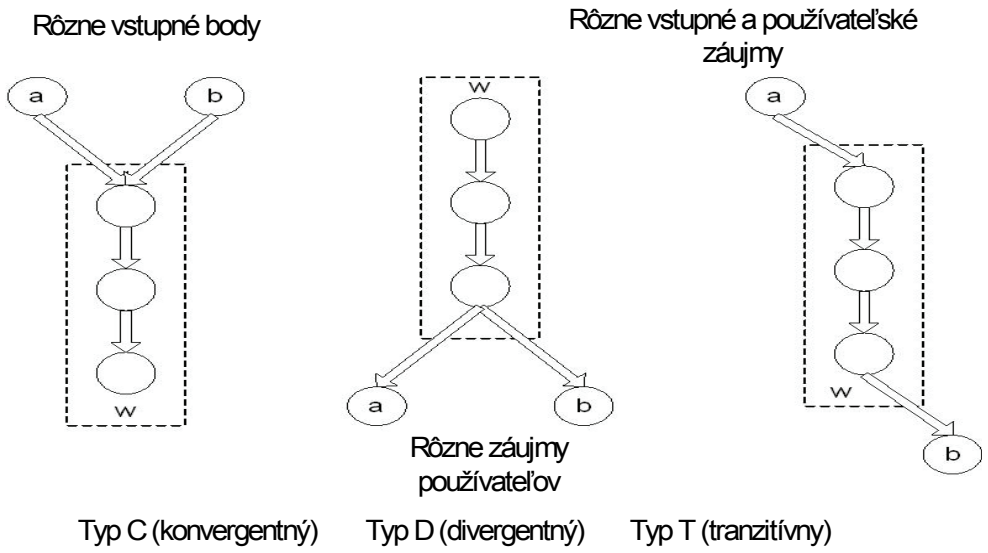
Definícia: Nech *a* je neopakujúca sa položka v postupnosti $s_1 = a_1 \rightarrow a_2 \rightarrow \ldots \rightarrow a_n$, a *b* je neopakujúca sa položka v postupnosti $s_2 = b_1 \rightarrow b_2 \rightarrow \ldots \rightarrow b_n$, potom webové stránky *a* a *b* vytvárajú sekvenčnú nepriamu asociáciu cez neprázdnu mediátorovú sekvenciu w, ak s_1= aw, alebo s_1= wa, a s_2 = bw alebo s_2 = wb a sú splnené nasledujúce podmienky (Tan et al., 2002, Agrawal et al., 1993, Agrawal & Srikant, 1994, Brin et al., 1997b):

Podpora ({a,b})< t_s, (položková podpora)

Podpora $(s_1) \geq t_f$, a súčasne Podpora $(s_2) \geq t_f$ (mediátorová podpora)

Závislosť $(a,w) \geq t_d$, Závislosť $(b,w) \geq t_d$ (mediátorová závislosť)

Sekvenčné nepriame asociácie môžu byť rôzneho typu (pozri obrázok 2-3).



Obrázok 2-3. Typy sekvenčných nepriamych asociácií (Zhong et al., 2003).

2.2.5 Implementácia

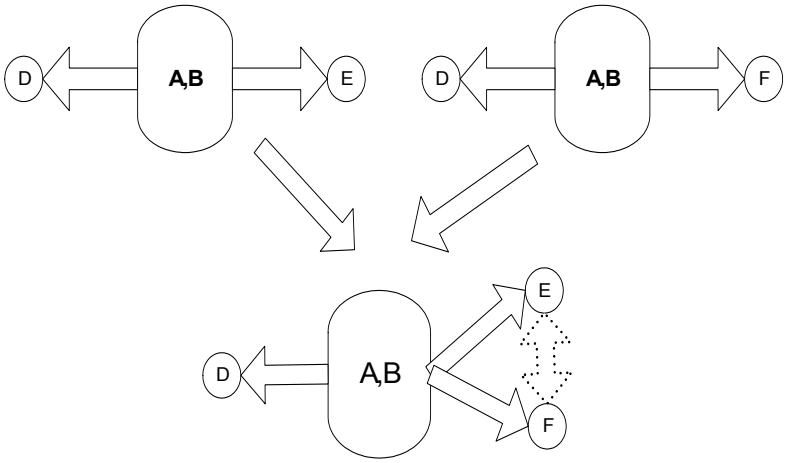
Algoritmus na získavanie nesekvenčných nepriamych asociácií opisuje nasledujúci príklad.

```
1. Extract the large itemsets L1,L2,...Ln, using standard mining
   algorithms.
2. P = Ø
3. for k = 2 to n do
4.   Ck+1← join (Lk,Lk)
5.   for each (a,b,M)∈ Ck+1 do
6.     if (sup({a,b}) < ts and d({a},M)≥td and d({b},M)≥td
7.       P = P ∪ (a,b,M)
8.   end
9. end
```

Príklad 2-1. INDIRECT algoritmus (Zhong et al., 2003).

Spájanie nepriamych asociácií

Nepriame asociácie možno zoskupiť do kompaktných štruktúr, ak sa delia o spoločný mediátor.



Obrázok 2-4. Spájanie nepriamych asociácií.

Pri výslednej asociácii treba kontrolovať stupeň asociácie medzi položkami E a F.

Experimentálne vyhodnotenie

Na dokázanie efektívnosti nepriamych asociácií sa testoval INDIRECT algoritmus na webovom serveri Minnesotskej Univerzity na oddelení počítačov (CS) a z priameho webu (ECOM). Množiny dát a nastavenie parametrov sú v tabuľke 2-1.

Tabuľka 2-1. Množiny dát a nastavenie parametrov pri experimente (Zhong et al., 2003).

Množiny dát	ts	tf	td	položky	Postupnosti
CS (Computer Science)	0,02	0,1	0,1	91443	34526
ECOM (online Web store)	0,005	0,05	0,2	6664	143604

V experimente sa zistilo približne 7500 nesekvenčných nepriamych asociácií <a,b,M> generovaných v rôznych krokoch algoritmu, viac ako 5500 nesekvenčných nepriamych spojení (a,b). Vykonali sa viaceré experimenty s rôznym nastavením parametrov.

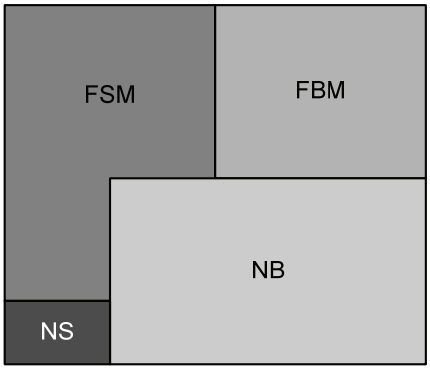
Hľadanie zaujímavých asociácií

Položky v sledovanej množine položiek si označme takto:

- FSM = frekventované položky s mediátorom
- FBM = frekventované položky bez mediátora
- NS = nefrekventované položky s mediátorom
- NB = nefrekventované položky bez mediátora

Ak NS/FSM = NB/FMB potom nepriama asociácia nie je prekvapujúca.

Ak je podiel NS/FSM malý a podiel NS/NB malý, potom je nepriama asociácia zaujímavá (pozri obrázok 2-5).



Obrázok 2-5. Hľadanie zaujímavých asociácií.

2.2.6 Zhodnotenie

V tejto časti sme opísali nové techniky hľadania nepriamych asociácií na webových dátach. Uviedli sme spôsoby hľadania zaujímavých asociácií. Nepriame asociácie spájajú spolu vzory, ktoré majú podobné subštruktúry. Spoločné subštruktúry (mediátory) často obsahujú navigačné stránky. Nepriame asociácie poskytujú alternatívny prístup na zachytenie zaujímavých nefrekventovaných vzorov.

Pre webové dáta predstavujú nepriame asociácie rôzne záujmy používateľov webu, ktorí zdieľajú podobné navigačné cesty. Tieto vzory sa nemôžu jednoducho objaviť pri použití štandardných asociácií a zhlukovacích techník. Nepriame asociácie sa môžu použiť na zhlukovanie vzorov do kompaktnejších štruktúr.

2.3 Znalostná indukcia obaľovačov určených na získavanie informácií z webu

Extrakcia informácií je proces identifikovania určitých fragmentov dokumentu, ktoré tvoria jeho hlavný sémantický obsah. Systémy realizujúce extrakciu informácií často závisia od extrakčných pravidiel prispôsobených konkrétnemu informačnému zdroju, aby zvládli štrukturálnu rôznorodosť typickú pre mnohé zdroje. Takéto systémy nazývame obaľovače (z angl. *wrapper*). Obaľovač sa definuje ako program alebo pravidlo, ktoré rozumie informáciám poskytovaným špecifickým zdrojom a prekladá ich do štandardnej formy tak, aby ich mohli použiť iné agenty.

Obaľovač sa špecializuje na určitý informačný zdroj. Rôzne stránky obsiahnuté na webe majú spravidla rôznu štruktúru, preto treba pre každú z nich definovať vlastný obaľovač. Na obrázku 2-6 sa nachádzajú ukážky troch stránok realitných kancelárií ponúkajúce domy na predaj. Všetky obsahujú podobné informácie, no každá z nich ich prezentuje v rôznej štruktúre (rôzne poradie, rôzne formátovanie). Pre každú z týchto stránok treba definovať vlastný obaľovač, ktorý sa špecializuje práve na konkrétnu štruktúru stránok. V pravej časti obrázka sa nachádzajú vydolované informácie z ukážok v jednotnej forme tak, ako ich potrebujeme na ďalšie spracovanie.

Obaľovače možno rozdeliť do troch kategórií na základe spôsobu ich vytvorenia:

- *Manuálne generované* – sú naprogramované priamo človekom. Keďže pre každú stránku je potrebný nový obaľovač, je potrebný ľudský zásah pri tvorbe

obaľovača pre každú stránku. Takisto zmeny v štruktúre stránok si vyžadujú manuálne vykonanie zmeny v obaľovači.

- *Heuristicky indukované* – vytvorené strojovo na základe všeobecne platných pravidiel. Napríklad znak $ a za ním nasledujúce číslo sa môže vyhodnotiť ako suma v USD. Ich nevýhodou je malá množina všeobecne platných pravidiel.
- *Znalostne indukované* – vytvorené strojovo na základe dopredu definovanej bázy znalostí platných pre určitú doménu. Bázu znalostí spravidla vytvára priamo človek. Výhodou tohto prístupu je možnosť zadefinovať jednu bázu znalostí a na jej základe vytvoriť veľa obaľovačov pracujúcich v rovnakej doméne.

Obrázok 2-6.Ukážka rôznych informačných zdrojov a informácií z nich získaných.

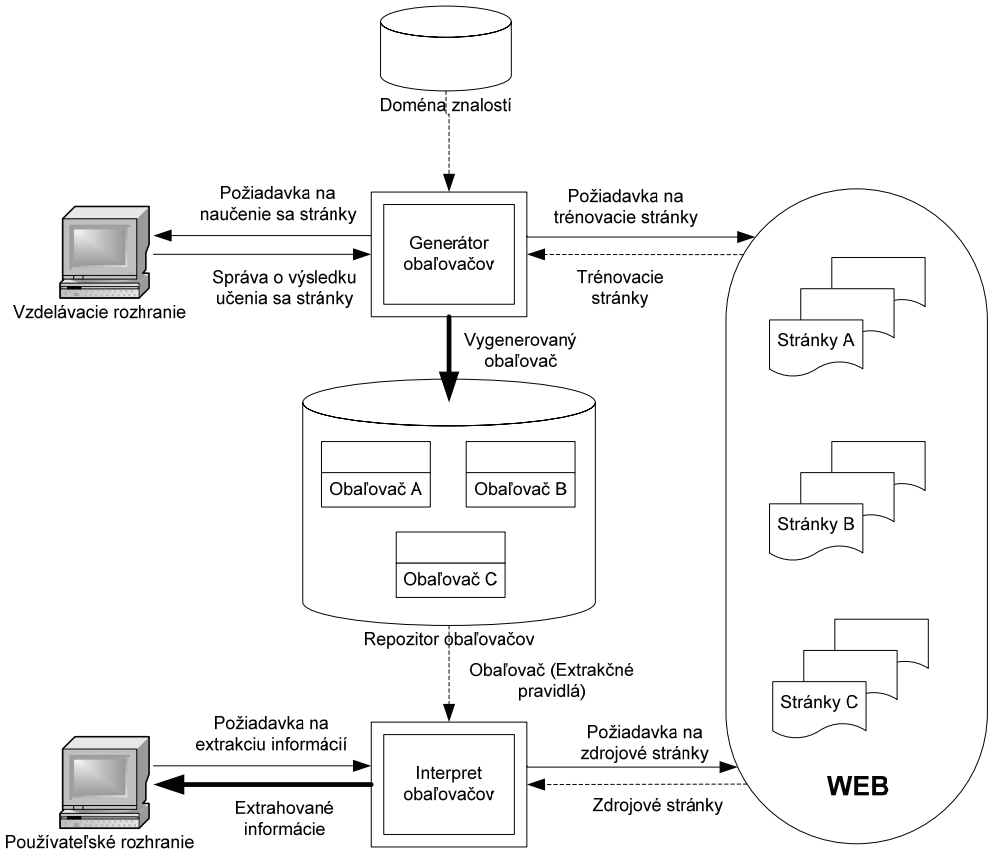
2.3.1 XTROS

Za príklad znalostného generátora obaľovačov možno považovať systém XTROS. Okrem automatického generovania obaľovačov dokáže s ich pomocou aj extrahovať informácie z korešpondujúcich zdrojov, takže ho možno považovať za komplexné riešenie v danej problematike.

V systéme XTROS sú jednotlivé obaľovače a aj báza znalostí definované ako XML dokumenty. Cieľom tohto prístupu bolo dosiahnuť vyššiu modulárnosť systému a umožniť ich modifikáciu aj pomocou externých nástrojov. Jeho výhodou je možnosť jednoduchej výmeny znalostnej bázy. Vďaka tomu sa XTROS stáva univerzálnym nástrojom, ktorý možno použiť v ľubovoľnej doméne, stačí len definovať znalostnú bázu pre túto doménu.

Princíp činnosti celého systému je znázornený na obrázku 2-7. V hornej časti obrázka sa nachádza generátor obaľovačov. Ten na pokyn používateľa dokáže s pomocou bázy doménových znalostí a sady trénovacích stránok vytvoriť obaľovač pre túto sadu. Uloží ho do úložiska obaľovačov.

V spodnej časti obrázka sa nachádza interpret obaľovačov. Jeho cieľom je na základe existujúceho obaľovača extrahovať informácie z prislúchajúcich stránok.



Obrázok 2-7. Schéma činnosti systému XTROS.

XTROS je určený na spracovávanie poloformálnych dokumentov, najmä značkovaných. V značkovaných dokumentoch každá časť dát, ktorá sa má extrahovať, sa vyjadrí pomocou páru značka-hodnota, kde značka definuje význam hodnoty. Ako príklad možnou použiť ukážky stránok realitných kancelárií z obrázku 2-6. Ide o pološtruktúrované stránky, nakoľko informácie o všetkých domoch sa prezentujú v rovnakej štruktúre ako na obrázku. Okrem toho ide značkovaný dokument, keďže každá informácia o dome sa reprezentuje dvojicou značka-hodnota: napríklad prvá ukážka obsahuje páry ako $3195000, 5 BR, 5 BA a pod. V tomto prípade môžu byť $, BR a BA rozpoznané ako značky vyjadrujúce množstvo v dolároch, počet spální a kúpeľní a 3195000 a 5 ako hodnoty. Všimnime si, že značka sa môže vyskytnúť pred aj za hodnotou.

2.3.2 Báza znalostí

Báza znalostí systému XTROS opisuje pojmy, na ktoré sa bude pri extrakcií informácií zameriavať. Ide o XML dokument, ktorého ukážka pre doménu predaja nehnuteľností sa nachádza v príklade 2-2.

```
<KNOWLEDGE>
     <OBJECTS>
            <OBJECT>PRICE</OBJECT>
            <OBJECT>BED</OBJECT>
            <OBJECT>BATH</OBJECT>
            <OBJECT>CITY</OBJECT>
            <OBJECT>MLS</OBJECT>
            <OBJECT>DETAIL</OBJECT>
            <OBJECT>IMG</OBJECT>
     </OBJECTS>
     <PRICE>
            <ONTHOLOGY>
                   <TERM>PRICE</TERM>
                   <TERM>$</TERM>
            </ONTHOLOGY>
            <FORMAT>
                   <FORM>[ONTOLOGY]DIGITS</FORM>
            </FORMAT>
     </PRICE>
     <BED>
            <ONTHOLOGY>
                   <TERM>BEDROOMS</TERM>
                   <TERM>BEDROOM</TERM>
                   <TERM>BEDS</TERM>
                   <TERM>BED</TERM>
                   <TERM>BR</TERM>
                   <TERM>BD</TERM>
            </ONTHOLOGY>
            <FORMAT>
                   <FORM>[ONTOLOGY]DIGITS</FORM>
                   <FORM>DIGITS[ONTOLOGY]</FORM>
            </FORMAT>
     </BED>
     <BATH>   ... </BATH>
     <MLS>    ... </MLS>
     <DETAIL> ... </DEATIL>
     <IMG>    ... </IMG>
</KNOWLEDGE>
```

Príklad 2-2. Ukážka XML reprezentácie bázy znalostí.

Obsah bázy znalostí sa definuje v XML elemente <KNOWLEDGE>. Ten obsahuje elementy <OBJECT>, v rámci ktorých sa nachádzajú jednotlivé pojmy bázy znalostí, napríklad PRICE (cena nehnuteľnosti), BED (počet spálni) a pod. Za nimi sa nachádzajú elementy, ktoré opisujú podrobnejšie jednotlivé pojmy, napríklad elementy <PRICE>, <BED> a pod. Opis týchto pojmov sa definuje dvoma elementmi: <ONTOLOGY> a <FORMAT>. Element <ONTOLOGY> obsahuje zoznam reťazcov, ktoré môžu predstavovať konkrétne značky v dokumente pre daný pojem. Napríklad počet spálni (BED) môže byť v dokumente označený ako BEDROOMS, BEDROOM, BEDS, BR alebo BD. Element <FORMAT> obsahuje informácie o type údajov (či ide o čísla alebo reťazce) a poradie, v akom sa hodnota vyskytuje so značkou. V prípade počte spálni je hodnotou číslo, ktoré sa môže vyskytovať pred alebo za značkou.

2.3.3 Generovanie obaľovača

Základnou funkciou generátora obaľovačov v doméne realít je naučenie sa formátu opisu domu. Vo všeobecnosti môžeme povedať, že stránka obsahujúca opisy domov pozostáva z hlavičky, tela a päty. Telo je sekvencia skúmaných domov, ktoré pozostávajú z atribútov. Úlohou obaľovača je identifikovať jednotlivé domy a jeho atribúty. Preto základom práce obaľovača je oddelenie hlavičky a päty od tela stránky. Systém XTROS to rieši priamou identifikáciu jednotlivých skúmaných entít, nie identifikáciu hlavičky a päty, ktoré treba odstrániť.

Ďalej opíšeme postup generovanie obaľovača. Vstupom pre tento proces je báza znalostí a sada trénovacích stránok.

Konvertovanie na logické riadky

Prvým krokom je rozdelenie HTML kódu trénovacích stránok na logické riadky. Logický riadok je totožný s riadkom, ktorý vidí používateľ pri prezeraní stránky vo svojom prehliadači. Okrem toho sú z HTML kódu odstránené všetky formátovacie značky s výnimkou `<a href = ..>` a `<img src = ..>`. Tým sa očistí zdrojový kód od nepotrebných častí a pripraví sa tak na ďalšie spracovanie.

Určenie významu logických riadkov

Druhou fázou algoritmu je určenie významu logických riadkov, pričom sa v nich vyhľadávajú objekty z bázy znalostí. Kontroluje sa, či riadok neobsahuje niektorý z reťazcov definovaných v elementoch `<TERM>` nachádzajúcich sa v elementoch `<ONTOLOGY>`. Súčasne sa zisťuje, či nájdený výskyt zodpovedá prislúchajúcim pravidlám formátovania definovaných v elemente `<FORMAT>`.

Ak dôjde k úspešnému rozpoznaniu pojmu z ontológie, tento pojem sa uloží do tabuľky pozostávajúcej z týchto stĺpcov: `object`, `line`, `cat`, `type` a `format`. `Object` opisuje význam riadku, `line` obsahuje pôvodný logický riadok spolu s identifikovaným pojmom z bázy znalostí, `cat` obsahuje identifikačné číslo daného objektu, `type` opisuje typ identifikovaných dát a `format` zachytáva poradie značky z bázy znalostí a hodnoty. Priradenie identifikačných čísel k pojmom z bázy znalostí definovanej v predchádzajúcej časti je definované v tabuľke 2-2.

Tabuľka 2-2. Priradenie identifikačných čísel k pojmom z bázy znalostí.

Identifikačné číslo	Prislúchajúci pojem z bázy znalostí
0	PRICE
1	BED
2	BATH
3	CITY
4	MLS
5	DETAIL
6	IMG
9	GENERAL TEXT

Tabuľka 2-3 zachytáva ukážku zaznamenaných výskytov pojmov bázy znalostí v trénovacej stránke.

Tabuľka 2-3. Príklad tabuľky identifikovaných pojmov z logických riadkov.

	Object	Line	Cat	Type	Format
1	{[IMG]}	<A HREF=" "> {[<IMG[{ ALT=".."}]>]}	6	IMGURL	[ONTOLOGY] IMGURL
2	{[PRICE]}	{[[{$}]3195000]}	0	DIGITS	[ONTOLOGY] DIGITS
3	{[BED]}	; {[5[{BR}]]}	1	DIGITS	DIGITS [ONTOLOGY]
4	{[BATH]}	; {[5[{BA}]]}	2	DIGITS	DIGITS [ONTOLOGY]
5	{[MLS]}	; 5000sf ; {[[{ MLS ID:#}] P209731]}	4	DIGITS	ONTOLOGY] DIGITS
6	{[DETAIL]}	{[<A HREF=" "> [{View Property}]]}	5	URL	URL [ONTOLOGY]

Nájdenie vzoru s najčastejším výskytom

Po vytvorení tabuľky výskytov treba určiť najčastejšie poradie, v ktorom sa jednotlivé pojmy z bázy znalostí v trénovacích stránkach vyskytujú. Pri hľadaní najčastejšieho vzoru postupujeme v týchto krokoch:

- Vytvoríme zoznam všetkých možných vzorov vyskytujúcich sa v trénovacích stránkach. V každom vzore sa môže vyskytovať jeden pojem iba raz!
- Ohodnotíme všetky vzory. Ohodnocovacia funkcia je počet pojmov vo vzore * počet výskytov vzoru na stránke.
- Vyberieme vzor s najväčšou hodnotou ohodnocovacej funkcie. Ten sa v trénovacích stránkach vyskytuje najčastejšie.

Vytvorenie obaľovača

Posledným krokom je vytvorenie samotného XML dokumentu obaľovača. V predchádzajúcich krokoch sme získali všetky potrebné znalosti, teraz z nich môžeme spojiť do výsledného produktu.

Ukážka dokumentu obaľovača, ktorý vznikol z prezentovanej bázy znalostí sa nachádza v príklade 2-3. Obaľovač je definovaný v rámci elementu `<Wrapper>`. Ten pozostáva z dvoch elementov: `<Form>` a `<Home>`. Pre nás je podstatný iba obsah elementu `<Home>`. Ten pozostáva z pojmov bázy znalostí, ktoré sú v ňom uložené v poradí, ktoré je definované vzorom najčastejších výskytov (v našom prípade `<Img>`, `<Price>`, `<Bed>` a pod.) Každý element pojmu z bázy znalostí obsahuje ďalšie elementy, ktoré opisujú formu výskytu tohto pojmu v danom type stránok. Element `<Ontology>` obsahuje reťazec reprezentujúci značku, ktorá v danom dokumente definuje pojem z bázy znalostí. `<Ident>` obsahuje reťazec, ktorý sa rovnako vysky-

tuje vo forme značky, ale nie je definovaný v báze znalostí. <Format> definuje formát extrahovaných údajov a <Operation> definuje poradie značiek a hodnoty, v akom sa vyskytujú v sledovaných stránkach.

```
<Wrapper>
     <Form>
            ...
     </Form>
<Home>
<Img>
<Ontology>ALT="View Details"</Ontology>
<Ident>SRC</Ident>
<Format>URL</Format>
<Operation>Ontology*Ident*Format</Operation>
</Img>
<Price>
<Ontology>$</Ontology>
<Ident>NULL</Ident>
<Format>Digits</Format>
<Operation>Ontology*Format</Operation>
</Price>
<Bed>
<Ontology>BR</Ontology>
<Ident>NULL</Ident>
<Format>Digits</Format>
<Operation>Format*Ontology</Operation>
</Bed>
...
</Home>
</Wrapper>
```

Príklad 2-3.Ukážka XML reprezentácie obaľovača.

2.3.4 Interpretovanie obaľovača

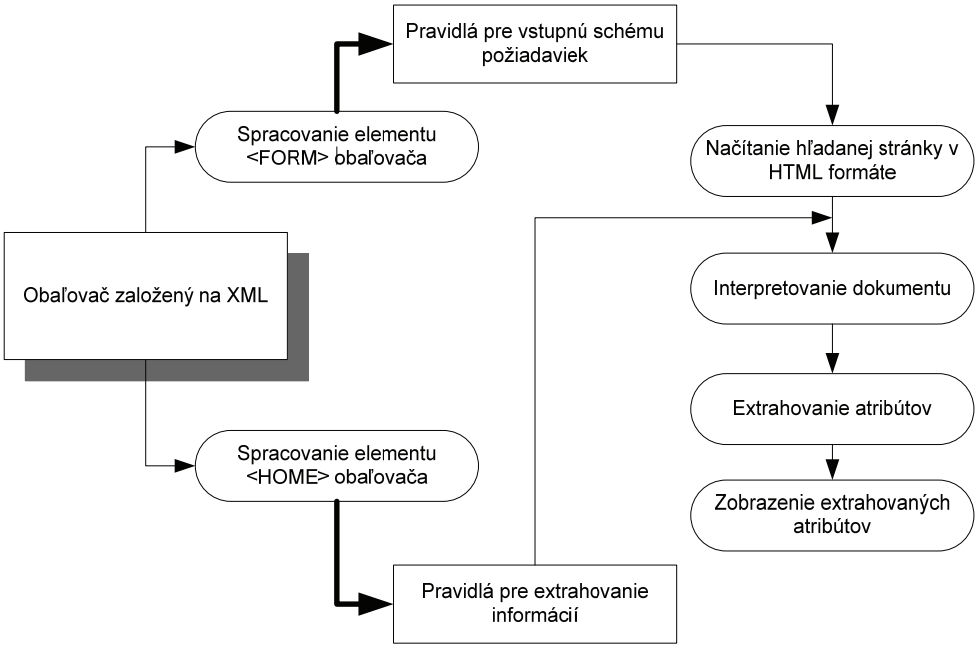
Systém XTROS obsahuje interpreter, ktorý na základe XML dokumentu obaľovača spracováva zdrojové stránky, výsledkom čoho je sada extrahovaných informácií. Činnosť interpretera zaznamenáva obrázok 2-8.

2.3.5 Implementácia a overenie riešenia

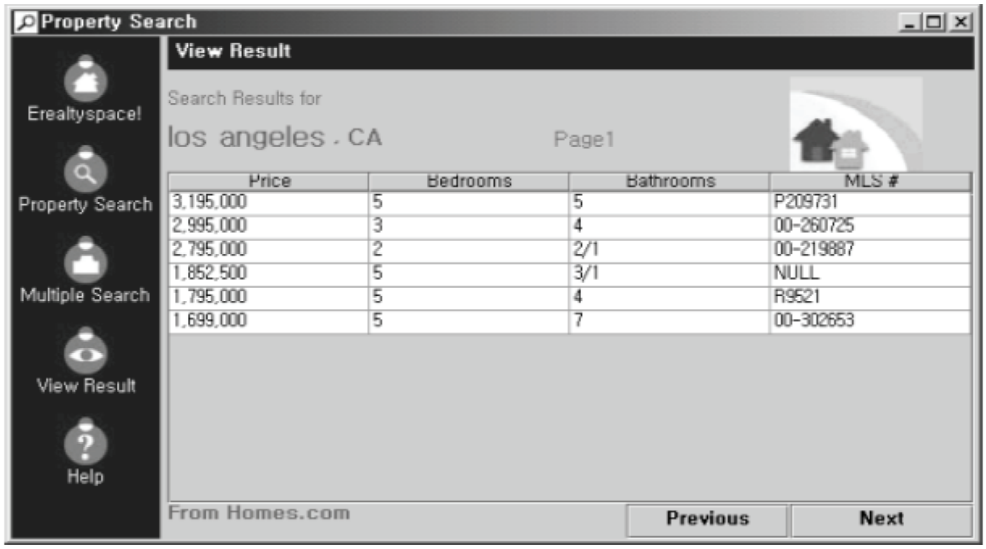
Systém XTROS úspešne implementovali v jazyku JAVA. Na obrázku 2-9 sa nachádza ukážka obrazovky systému. V danom okne sa prezentujú výsledky úspešnej extrakcie informácií zo stránky realitnej kancelárie.

Testovanie prebehlo s touto množinou stránok:

- Homes (http://www.homes.com)
- Realtor (http://www.realtor.com)
- Yahoo Real Estate (http://realestate.yahoo.com)
- iOwn (http://www.iown.com)
- CyberHomes (http://www.cyberhomes.com)
- HomeScape (http://www.listinglink.com)



Obrázok 2-8. Schéma činnosti interpretera obaľovača.



Obrázok 2-9. Obrazovka systému XTROS (Zhong et al., 2003).

- Move (http://www.move.com)
- ERA Real Estate (http://www.era.com)
- Coldwell Banker (http://www.coldwellbanker.com)
- HomeSeekers (http://www.homeseekers.com)

XTROS dokázal vygenerovať obaľovače pre 8 z 10 stránok. Pre 2 z nich nebolo generovanie úspešné, nakoľko informácie na stránkach sa neposkytovali vo forme značkovaných dokumentov, ale vo forme tabuľky. Výsledky extrahovania dosiahli pri väčšine 100% úspešnosť. Problémy robili najmä neštandardné prípady, napríklad:

- „6 (full) 1(half) Baths“ – záznamy neboli vo formáte čísel, ako sa očakávalo,
- „- Beds, - Baths“ – záznamy neobsahovali žiadne relevantné hodnoty,
- „1 Bath“ – značka na extrahovanej stránke sa odlišovala od značky z testovacích stránok (obaľovač očakával „Baths“).

2.3.6 Zhodnotenie

V práci opisujeme informačný extrakčný systém XTROS, ktorý reprezentuje doménové znalosti a obaľovače pomocou XML dokumentov a automaticky generuje obaľovače z poloformálnych značkovaných dokumentov. XTROS ukázal dobrý výkon na viacerých reálnych webových stránkach zaoberajúcich sa problematikou nehnuteľností, no možno ho upraviť na prácu v ľubovoľnej doméne vďaka možnosti jednoduchej zmeny bázy znalostí definovanej pomocou XML.

Jedným z nedostatkov systému XTROS je jeho závislosť na značkovaných dokumentoch. Preto sa jeho autori rozhodli rozšíriť jeho spracovanie aj o iné typy dokumentov a napríklad tabuľky.

2.4 Dolovanie vo webových záznamoch

Webová inteligencia je oblasť, v ktorej prebieha intenzívny výskum. Rýchlo sa rozvíjajúca svetová pavučina WWW je zdrojom pre hľadanie nových výziev, ako aj príležitostí.

Na web sa môžeme zjednodušene pozerať ako na zbierku veľkého množstva vzájomne prepojených dát, informácií a zdrojov znalostí. Z dôvodu veľkej odlišnosti zdrojov na webových stránkach je nutné navrhnúť a implementovať rôzne typy webových informačných systémov, ako aj informačné systémy na špeciálne účely, akými sú napríklad: inteligentné vyhľadávače, inteligentné webové agenty a inteligentné webové prehliadače.

Webové záznamy zaznamenávajú aktivitu používateľov a využitie zdrojov webových stránok počas toho, čo používateľ tieto stránky prezerá. Tieto záznamy sú preto primárnym zdrojom údajov, ktoré je možné analyzovať pri úlohách dolovania požadovaných znalostí.

2.4.1 Prehľad dolovania vo webových zdrojoch

Dolovanie v dátach sa zaoberá extrakciou alebo dolovaním užitočných znalostí z veľkého množstva dát uložených v databázach alebo iných zdrojoch informácií.

S rýchlym nárastom množstva informácií na webových stránkach získavame množstvo rôznych zdrojov dát, ktoré môžeme skúmať prostredníctvom dolovania v dátach. Na automatické objavovanie a extrakciu webových informácií z webových záznamov sa môžu použiť techniky dolovania v dátach.

Klasifikácia dolovania vo webových zdrojoch

Na základe funkcionality a aplikačnej domény systémov na dolovanie vo webových zdrojoch navrhli množstvo rôznych klasifikácií úloh dolovania:

- *Lokalizácia zdroja* – nájdenie požadovaných dokumentov, informácií a služieb.
- *Extrakcia informácií* – automatická extrakcia želaných informácií z objavených zdrojov.
- *Generalizácia* – nájdenie všeobecných vzorov na individuálnych webových stránkach, taktiež aj cez viacero stránok.
- *Analýza* – validácia a/alebo interpretácia objavených všeobecných vzorov.

Na základe prístupných dát môžeme dolovanie kategorizovať podľa typu dolovaných dát: dolovanie vo webovom obsahu, dolovanie štruktúry a využitia webových stránok.

- *Obsah* – dáta predstavujúce webové stránky a vyjadrujúce informácie pre používateľov, napríklad: html, grafické dáta, audio súbory na webovej stránke.
- *Štruktúra* – dáta koncipujúce hyperlinkovú štruktúru webovej stránky, napríklad: rôzne HTML značky použité pri odkazovaní jednej stránky na inú stránku.
- *Využitie* – dáta reflektujúce využitie zdrojov webu, napríklad: položky histórie vo webovom prehliadači, dočasné Internetové súbory, záznamy proxy servera a webového servera.
- *Profil používateľa* – dáta poskytujúce demografické informácie o používateľoch webovej stránky, napríklad: registračné dáta používateľov a informácie z profilov zákazníkov.

Dolovanie v obsahu webu

Dolovanie v obsahu webu opisuje objavenie užitočných informácií z webového obsahu/dát/dokumentov. Keďže tento obsah môže zahŕňať veľmi rozsiahle množstvá dát, je nutné aplikovať rôzne techniky na dolovanie v obsahu webu. Množstvo metód dolovania v obsahu webu je založených na neštruktúrovaných textových dátach, resp. semištruktúrovaných HTML dokumentoch.

Viac špecifické oblasti dolovania v obsahu webu sú dolovanie v obsahu textových dát, dolovanie v obsahu textov a kategorizácia textov. Vo všeobecnosti môžeme rozdeliť dolovanie v obsahu webu na získavanie informácií a skúmanie databáz. Cieľom dolovania v obsahu webu z pohľadu získavania informácií je najmä asistovať alebo napomáhať pri vylepšovaní hľadania informácií pre používateľov. Výsledok dolovania môžeme aplikovať vo webových vyhľadávačoch, webových personalizovaných alebo odporúčacích systémoch a adaptívnych webových rozhraniach.

Dolovanie štruktúry webu

Dolovanie webovej štruktúry sa snaží objaviť model podstaty hyperlinkovej štruktúry webových stránok. Model reflektuje topológiu hyperlinkových obsahov týchto stránok. Webové stránky možno klasifikovať na *authority* alebo *hub* stránky a informácie generovať na základe podobnosti, resp. rozdielnosti medzi rôznymi webovými stránkami.

Dolovanie využitia/záznamov webu

Záznamy webového servera predstavujú veľké množstvo explicitných požiadaviek používateľov na zdroje webovej stránky na strane servera. Predstavujú hlavný zdroj pre

dolovanie využitia webu, resp. dolovanie vo webových záznamoch. Iné webové záznamy, ako záznamy proxy servera, histórie webového prehliadača a *cookies* súbory, tiež zaznamenávajú požiadavky používateľov. Množstvo štúdií o dolovaní využitia webu a dolovaní vo webových záznamoch je založených práve na dolovaní v dátach z webového servera.

Kombinácia dolovania v obsahu, dolovania štruktúry a využitia webu

Vo veľa prípadoch je obsah webu, štruktúra a informácie o využití reprezentovaná v tom istom dátovom súbore. Napríklad meno súboru, ktoré sa objaví v záznamových súboroch a dátach štruktúry webu, obsahuje užitočné informácie o obsahu. Mohli by sme napríklad predpokladať, že súbor pomenovaný „WebLogMining.html“ musí obsahovať informácie o dolovaní v záznamoch webu. O troch kategóriách dolovania vo webových záznamoch nemôžeme uvažovať izolovane od ostatných – dolovanie v obsahu webu, napríklad niekedy musí využiť dáta zo štruktúry webu pre klasifikovanie webovej stránky.

2.4.2 Príprava dát

Presné a vhodné dáta sú kľúčové pre úlohy dolovania v dátach. To isté platí pre úlohy využitia webu/webového dolovania. Záznamy webového servera sú primárne zdroje dát pre dolovanie v dátach o využití webu a to aj napriek tomu, že dáta zozbierané webovým serverom sú nekompletné a nedostatočné. Pred aplikovaním techník dolovania dát treba aplikovať rozsiahle dátové predspracovanie a dátovú abstrakciu.

Zber dát

Záznamy webového servera zaznamenávajú požiadavky používateľov na webovú stránku na strane servera. Iné webové záznamy, ako záznamy proxy servera, história webového prehliadača a súbory *cookies* tiež zaznamenávajú požiadavky od používateľov tieto však mimo servera.

Záznamy webového servera – sú vo všeobecnosti veľké a väčšinou odkazujú na tri záznamové súbory, ako *access*, *referrer* a *agent* v závislosti na nastavení servera. *Access* záznamy obsahujú všetky položky o transakciách medzi webovým serverom a webovým prehliadačom. *Referrer* záznamy obsahujú informácie o stránke, z ktorej používateľ navštívil aktuálnu stránku. *Agent* záznamy obsahujú záznamy o webových prehliadačoch ktoré použili používatelia pri návšteve stránky.

Formát HTML dokumentu určuje, že *access* záznam pre aplikáciu dolovania vo webových dátach vždy obsahuje aj redundantné dáta. HTML webová stránka zvyčajne obsahuje veľa súborov. Okrem toho HTML súbor môže obsahovať aj odkazy na grafické, video a audio súbory. Keď používateľ zadá požadovanú URL adresu, alebo keď klikne na niektorý odkaz v rámci stránky, väčšinou sa pre webový prehliadač odosiela viacero súborov v poradí, v akom sú požadované. Kedykoľvek, keď sa odošle súbor pre webový prehliadač, sa aktualizujú príslušné záznamové súbory.

V závislosti od nastavenia softvéru na webovom serveri sa môžu zaznamenať údaje jednoduché alebo zložité, a taktiež sa môžu zaznamenať v rôznom tvare. Vo všeobecnosti záznamy servera obsahujú tieto položky:

- čas a dátum transakcie,
- meno a veľkosť odoslaného súboru,
- Internetová adresa, na ktorú bol súbor odoslaný,

- ako bol súbor odoslaný,
- možný odkaz, ktorý viedol na požadovanú stránku,
- informácia o webovom prehliadači,
- výsledok prenosu,
- identifikátor používateľa (ak je to možné).

Množstvo webových serverov, ako napríklad Apache a ISS umožňujú administrátorovi nastaviť vlastný formát zaznamenávania, a taktiež umožňujú zaznamenávanie do jedného súboru vo formáte *Combined Log Format*. Použitie proxy serverov, resp. lokálnej *cache* môže značne ovplyvniť relevantnosť údajov zaznamenaných na strane servera.

Záznamy zo strany klienta – kolekcia dát na strane klienta môže vyriešiť problémy zapríčinené proxy serverom, resp. lokálnou *cache*. Súbor s históriou navštívených stránok ako aj dočasné internetové súbory sú súbory lokálnej *cache,* ktoré generuje webový prehliadač na strane klienta. Používajú sa z dôvodu zníženia sieťovej réžie pri opakovanom prístupe k webovým stránkam. Súbor s históriou zaznamenáva všetky stránky navštívené používateľom ako aj čas návštevy stránky. Dáta uložené v súbore histórie a v dočasných internetových súboroch vo všeobecnosti odrážajú aktivitu jedného konkrétneho používateľa. Tento predpoklad je vo všeobecnosti príliš reštriktívny na to, aby mohol byť použitý vo všeobecnosti. Keďže jeden počítač môže používať viacero používateľov, na ich identifikáciu je nutné použiť *cookies*.

Okrem použitia *cookies* je možné zmenou zdrojového kódu prehliadačov použiť vzdialených agentov (Java script alebo Java applety). Hlavnou nevýhodou tohto riešenia je však neochota jeho použitia zo strany používateľov z obavy narušenia svojho súkromia.

Záznamy z proxy servera – proxy server slúži ako úroveň medzičlánku *cache* medzi webovým prehliadačom klienta a webovým serverom. Jeho hlavným účelom je znížiť čas načítania webových stránok, ako aj zníženie sieťového zaťaženia. Proxy server tiež spôsobuje, že množstvo používateľov, ktorí používajú ten istý proxy server, sa v záznamoch servera objavuje ako jeden používateľ. Toto môže spôsobovať problémy v identifikácii používateľa a identifikácii sedenia (*session*) počas dátového predspracovania.

Predspracovanie dát

Hlavnou úlohou predspracovania dát je transformácia zaznamenaných záznamov do použiteľnej formy pre neskoršie použitie. S týmto cieľom súvisí najmä prečistenie a kompresia dát.

Prečistenie dát vymaže také položky webových záznamov, ktoré sú irelevantné pre úlohy dolovania v dátach. V štandardnom prípade sa mažú dva typy dát. Prvý typ sú položky zaznamenané z dôvodu prístupu ku grafickým, zvukovým, video a množstvu iných multimediálnych súborov. Ďalším typom položiek na zmazanie sú položky generované webovými agentami, resp. webovými pavúkmi. Taktiež je možné zo záznamov vylúčiť niektoré typy súborov ako aj metódy transferu GET/POST/HEAD.

Abstrakcia dát

Abstrakcia dát používa heuristiku na separovanie dát do sémanticky významných konceptov a fráz. Medzi najčastejšie používané dátové abstrakcie patria: identifikácia používateľa, identifikácia sedenia a kompletizácia cesty.

Identifikácia používateľa – identifikácia používateľa nie je jednoduchá úloha, keďže používateľ môže na prístup použiť rôzne počítače, resp. rôzne prehliadače v rámci jedného počítača. Proxy server spôsobuje problém, že všetci používatelia využívajúce služby daného proxy servera sa na strane koncového servera javia ako jeden používateľ. Najjednoduchším riešením tohto problému je spoľahnúť sa na implicitnú a explicitnú kooperáciu používateľa. Taktiež je možné použiť rôzne modifikované prehliadače, resp. Java applety, ktoré môžu pomôcť pri identifikácii používateľa.

Ak chceme identifikovať používateľa na základe záznamov webu, treba použiť heuristické pravidlá. Je napríklad rozumné usudzovať, že rôzny typ agenta pre konkrétnu IP adresu reprezentuje rôzneho používateľa. Ďalšou možnosťou pre určenie konkrétneho používateľa je použiť kombináciu záznamov *access*, *referrer* a topológie stránky. V závislosti na rôznych situáciách a aplikáciách je nutné použiť rôzne heuristické metódy.

Identifikácia sedenia – za sedenie môžeme považovať množinu kliknutí používateľa krížom cez viaceré stránky. Sedenie môžeme definovať napríklad kombináciou IP adresy, id používateľa, URL adresou ako aj časom prístupu. Jednoduchšou metódou je definovanie časového okna, v rámci ktorého je akcia používateľa považovaná za jedno sedenie. V prípade, že používateľ počas prezerania prekročí časový limit sedenia (väčšinou 30 min.), považuje sa za ďalšieho používateľa.

Identifikácia transakcie – cieľom identifikácie transakcie je vytvorenie zmysluplných zhlukov referencií pre každého používateľa. Identifikácia transakcie je dôležitá pre asociačné dolovanie. Heuristické metódy, ktoré sa používajú na identifikáciu transakcie sú napríklad:

- identifikácia transakcie referenčnou dĺžkou,
- identifikácia transakcie maximálnou spätnou referenciou,
- identifikácia transakcie časovým oknom.

2.4.3 Dolovanie v dátach a analýza vzorov

Algoritmy pre dolovanie v dátach možno aplikovať na objavenie zaujímavých a neznámych vzorov s predpracovanými dátami uloženými v záznamoch webového servera. Výsledky dolovania v záznamoch webu sa môžu pohybovať v intervale jednoduchých štatistických informácií až po viac komplikované znalosti, akými sú napríklad klasifikačné pravidlá, asociačné pravidlá a sekvenčné prístupové vzory.

Štatistická informácia

Štatistické metódy predstavujú primárny prístup použitý v skorších systémoch na dolovanie v dátach. Často sa používajú v komerčných softvéroch na analýzu záznamov webu na poskytnutie sumárnych správ o zdrojoch prístupu na webovú stránku. Sumárne správy väčšinou zahŕňajú počet celkových návštev, počet chýb, ktoré sa vyskytli, celkový počet prenesených slabík a podobne.

Asociačné pravidlá

Dolovanie asociačných pravidiel sa pôvodne zaviedlo a študovalo v kontexte transakčných databáz. Zaoberá sa problémom hľadania podmnožín položiek zvaných *itemsets*, ktoré často a súčasne kupujú zákazníci. Keď použijeme dolovanie asociácií na dolovanie vo webových záznamoch, môžeme objaviť koreláciu medzi webovými stránkami a zaujímavými prístupovými vzormi.

Klasifikácia

Klasifikácia je dôležitý problém v oblasti dolovania v dátach. V klasifikácii sa štandardne používa množina pozitívnych a negatívnych vzorov ako trénovacie dáta. Cieľom je odvodiť model alebo opis pre každú triedu. Model môže byť použitý na klasifikáciu budúcich možností, ktorých trieda je neznáma. Populárnym algoritmom na klasifikáciu je triedenie pomocou rozhodovacích stromov.

Klasifikácia nám môže pomôcť pri vytváraní profilov používateľov, klasifikácii používateľov a obsahu webu.

Vyhodnotenie vzorov a pravidiel

Cieľom analýzy vzorov, pravidiel a vyhodnocovania je vyfiltrovanie nezaujímavých pravidiel alebo vzorov z množiny nájdenej vo fáze objavovania vzorov a pravidiel. Doménové znalosti predstavujú veľmi dôležitý faktor pre elimináciu nepoužiteľných a nezaujímavých vzorov a pravidiel.

2.4.4 Aplikácia

Elektronický obchod je jedným z hlavných hnacích síl poháňajúcich rýchly vývoj webových stránok. Výsledky z analýzy záznamov webu/použitia webu môžu poskytnúť príležitosti pre elektronický obchod v rôznych aspektoch. Môžu byť použité na zvýšenie efektívnosti a účinnosti týchto systémov a vylepšiť funkcionalitu, služby a jednoduché použitie inteligentných webových informačných systémov.

Napriek tomu, že niektoré z algoritmov dolovania v dátach sú aplikačne nezávislé, je efektívne použitie dolovania v dátach doménovo a aplikačne silne závislé.

Web predpríprava

Web predpríprava zahŕňa predprípravu zo servera pre klienta ako aj predprípravu z disku servera do pamäte servera. Pri napomáhaní v rozhodovaniach o predvýbere sa môžu použiť zhluky stránok a dolovanie sekvenčných vzorov prostredníctvom dolovania webu.

Lepší dizajn a organizácia webových stránok

Kvalita webovej stránky ako aj jej organizácie a prezentácie sa môžu vyhodnotiť práve vďaka analýze prístupu používateľov k webovej stránke. Veľký rozsah návštevnosti odráža vysoký záujem používateľov o webovú stránku. Kvalitu webovej stránky môže ovplyvniť množstvo faktorov akými sú obsah, prezentácia, jednoduché použitie, čas odozvy a podobne. Výsledky dolovania v záznamoch webu sa môžu použiť pre zlepšenie dizajnu webovej stránky ako aj zvýšenie spokojnosti používateľov.

Personalizácia webových stránok

Množstvo personalizovaných a odporúčacích webových systémov závisí od profilov používateľov. Poznáme dva rôzne spôsoby pre vytvorenie profilov používateľov: priama metóda prostredníctvom explicitnej kooperácie používateľa a webového agenta a nepriama metóda pomocou dolovania v dátach. Priama metóda má limitovanú aplikovateľnosť a môže byť invazívna a nepríjemná pre používateľa. Nepriame metódy môžu byť neinvazívne a súčasne anonymné pre používateľa. Táto metóda tiež zabraňuje slabému výkonu systému spôsobeného vzdialenými agentami.

Adaptívne webové stránky

Adaptívne webové stránky súvisia s personalizáciou webu a odporúčaním zdrojov na webe. Modifikácia adaptívnych webových stránok a ich generovanie sa venuje automatickej modifikácii obsahu a organizácie webových stránok použitím výsledkov dolovania využívania webu.

Inteligentné webové agenty

Inteligentné webové agenty založený na dolovaní vo webových záznamoch poskytujú množstvo služieb pre dizajnérov a používateľov webovej stránky, akými sú optimalizácia jej štruktúry, zvýšenie bezpečnosti, presonalizácia prezentácie, odporúčaní produktov a služieb a podobne. Poskytujú informácie pre dizajnérov webových stránok, čím je možné dosiahnuť lepšiu organizáciu webových stránok a tým zvýšiť spokojnosť používateľa.

2.4.5 Zhodnotenie

Webové záznamy sú dôležitým zdrojom dát pre dolovanie užitočných znalostí na zvýšenie účinnosti a efektivity webových informačných systémov. Dolovanie využitia webu/webových záznamov bude hrať rozhodujúcu rolu vo webovej inteligencii.

3 VYHĽADÁVANIE INFORMÁCIÍ NA WEBE A MANAŽMENT ZNALOSTÍ

Táto časť sa venuje moderným trendom vo vyhľadávaní informácií na webe a v iných informačných systémoch, zahŕňajúc v to aj prácu so znalosťami.

Vyhľadávanie na internete umožňujú automatické vyhľadávače (roboty, pavúky), ktoré prehľadávajú hierarchicky podľa obsahu sieť hypertextových odkazov a doménových záznamov. V informatike sa zaviedol pojem „web so sémantikou“, ktorý vyzdvihuje význam obsahu vo vyhľadávaných informáciách. Spracovanie obsahu na webe má byť automatizovateľné. Potom vyhľadávače namiesto hľadaných informácií ponúknu znalosti.

Systémy pre vyhľadávanie a spracovanie znalostí a všeobecne prácu so znalosťami označujeme ako manažment znalostí. Aby systém vedel so znalosťami správne zaobchádzať, musí ich vedieť kategorizovať, analyzovať, dalo by sa povedať, že rozumieť im.

3.1 Osobné a zamerané webové pavúky

Webový pavúk je program, ktorý prechádza informačným priestorom World Wide Web, pričom sleduje hypertextové odkazy a sťahuje webové dokumenty pomocou HTTP protokolu. Do tejto kategórie spadajú aj meta-vyhľadávacie pavúky (pavúky, ktoré sa pripájajú k iným vyhľadávacím službám a kombinujú ich výsledky). Iné webové roboty ako napr. shopboty, chatboty a chatterboty sa vo všeobecnosti nepovažujú za pavúky. V súčasnosti sa dá výskum webových pavúkov klasifikovať podľa týchto kategórií:

- *Rýchlosť a efektívnosť.* Táto kategória výskumu sa zameriava na zvyšovanie rýchlosti sťahovanie pavúkov. Projekty tejto kategórie sa zameriavajú na stavbu rýchlych pavúkov, ktoré sa dajú rozširovať aj na veľké zbierky dokumentov pomocou optimalizácií operácií ako napr. vstupno-výstupné procedúry, preklad IP adries.
- *Pravidlá prehľadávania.* Výskum v tejto kategórii sa zaoberá správaním pavúkov a ich vplyvom na ostatné systémy a WWW ako celok. Správne navrhnutý pavúk by nemal preťažovať webové servery. V súčasnosti existujú dva štandardy pre

ovplyvňovanie správania pavúkov autormi stránok: určovanie stránok, ktoré nemá pavúk sťahovať pomocou súboru *robots.txt* a určovanie toho, či sa má stránka používať v pavúkovi pre indexáciu alebo získavanie odkazov pomocou meta-značky. Napriek tomu, že tieto štandardy nie sú povinné, väčšina pavúkov ich dodržiava a riadi podľa nich svoje správanie.

- *Získavanie informácií.* Väčšia časť výskumu ohľadom pavúkov sa sústreďuje na túto oblasť. V rámci nej sa skúmajú rôzne algoritmy a heuristiky tak, aby pavúk získal požadované informácie z webu čo najefektívnejšie.

Štyri hlavné použitia pre webové pavúky sú:

- *Osobné vyhľadávanie.* Osobné pavúky sa snažia hľadať webové stránky zaujímavé pre používateľa. Keďže tieto pavúky sa väčšinou vykonávajú na klientskom počítači, majú k dispozícii väčší výpočtový výkon a poskytujú viac možnosti ako pavúky umiestnené na serveri.
- *Vytváranie zbierok.* Webové pavúky sa používajú na budovanie veľkých zbierok webových stránok, ktoré sa používajú na výpočet indexov každej vyhľadávacej služby. Ďalej sa tieto zbierky dajú použiť pri tvorbe lexikónov slov alebo zbieraní emailových adries.
- *Archivácia.* Niektoré projekty sa snažia robiť archiváciu určitých stránok, prípadne celého webu[3].
- *Webové štatistiky.* Z veľkého množstva webových stránok zozbieraných pomocou pavúkov sa dajú získať užitočné štatistiky o webe, ako napríklad priemerná dĺžka HTML dokumentu, priemerný počet odkazov, priemerný počet odkazov na neexistujúce stránky a ďalšie.

3.1.1 Analýza obsahu a štruktúry webu

Rôzne techniky analýz používaných v pavúkoch sa dajú rozdeliť do dvoch hlavných kategórií: analýzy založené na obsahu a analýzy založené na odkazoch.

Pri analýzach založených na obsahu sa pre odvodenie informácií o stránke analyzuje samotný HTML kód stránky. Napríklad z textového tela stránky sa dá zistiť, či je stránka relevantná pre hľadanú doménu. Pre získanie kľúčových konceptov opísaných na stránke sa dajú použiť indexovacie techniky. Ďalšie zlepšenie analýzy sa dá dosiahnuť použitím doménových znalostí.

Dôležitým zdrojom informácií o stránke je aj jej URL adresa, ktorá identifikuje adresu servera, doménové zaradenie webového servera (stránky v doméne gov môžu byť považované za dôležitejšie ako stránky v doméne com), hĺbku umiestnenia v adresárovej štruktúre webového servera (dôležité stránky nebývajú umiestnené hlboko v adresárovej štruktúre).

Druhá skupina analýz využíva pre získanie informácií odkazy medzi stránkami. Základný predpoklad použitý pri týchto analýzach je, že ak autor webovej stránky *A* umiestnil odkaz na webovú stránku *B*, tak si myslí, že stránka *B* je relevantná alebo podobná stránke *A* a je kvalitná. Pre odkaz, ktorý ukazuje na danú stránku, sa používa termín *vstupný odkaz*. Vo všeobecnosti platí, že o čo je stránka lepšia, o to viac má

[3] The Internet Archive, http://www.archive.org

vstupných odkazov. Naviac je vhodné priraďovať stránkam, na ktoré sa odkazujú iné kvalitné stránky, väčšiu prioritu. Takto sa správajú najznámejšie dva algoritmy pre analýzu odkazov PageRank (Brin & Page, 1998) a HITS (Kleinberg, 1997).

Algoritmus PageRank počíta pre každú stránku skóre, ktoré je váženým súčtom kvality stránok pripojených vstupnými odkazmi na danú stránku, pričom kvalita týchto odkazujúcich stránok sa taktiež určuje pomocou algoritmu PageRank. Rekurentný vzorec na výpočet skóre stránky *p* je potom:

$$PageRank(p) = (1-d) + d \cdot \sum_{\forall qukazujúcenap} \left(\frac{PageRank(q)}{c(q)} \right)$$

kde *d* je tlmiaci faktor v rozmedzí od 0 po 1 a *c(p)* je počet výstupných odkazov na stránke *q*. Intuitívne sa dá povedať, že stránka bude mať vysoké PageRank skóre, ak na ňu odkazuje veľa stránok tiež s vysokým PageRank hodnotením.

Algoritmus PageRank sa zakladá na modeli náhodného chodenia – skóre stránky je úmerné pravdepodobnosti, že používateľ, ktorý náhodne kliká na odkazy, sa dostane na túto stránku. Algoritmus sa úspešne používa vo vyhľadávacom systéme Google. Jeho hlavnou nevýhodou je vysoká výpočtová náročnosť.

HITS (hyper-link-induced topic search) algoritmus sa podobá algoritmu PageRank. V algoritme HITS sa stránky obsahujúce informácie relevantné k príslušnej problémovej oblasti alebo vyhľadávaciemu dotazu označujú ako autoritatívne stránky. Prepojovacie stránky nemusia byť autoritatívnymi stránkami, ale obsahujú odkazy na iné autoritatívne stránky.

Stránka, na ktorú sa odkazuje veľa iných stránok, by mala byť dobrou autoritatívnou stránkou. Stránka, ktorá obsahuje odkazy na veľa iných stránok, by mala byť dobrou prepojovacou stránkou. Na základe týchto myšlienok HITS algoritmus počíta dve skóre pre každú stránku:

$$AuthorityScore(p) = \sum_{\forall qodkazujúcenap} HubScore(q)$$

$$HubScore(p) = \sum_{\forall rodkazovanézp} AuthorityScore(r)$$

Na stránku s vysokým autoritatívnym skóre odkazuje veľa stránok s vysokým prepojovacím skóre a stránka s vysokým prepojovacím skóre odkazuje na veľa stránok s vysokým autoritatívnym skóre. Algoritmus HITS používa napríklad vyhľadávacia služba Clever. Podobne ako v prípade algoritmu PageRank sú nevýhodou HITS vysoké výpočtové nároky, lebo prepojovacie a autoritatívne skóre sa musia počítať v cykle.

3.1.2 Grafové prehľadávacie algoritmy

Tradičné grafové algoritmy sa skúmajú aj v odbore informatiky. Keďže na web sa dá pozerať ako na orientovaný graf s množinou uzlov (stránok) prepojených orientovaných hranami (odkazmi), dajú sa niektoré grafové algoritmy použiť vo webových aplikáciách.

Prvá kategória grafových prehľadávacích algoritmov obsahuje jednoduché algoritmy, ako napríklad hľadanie do šírky a hľadanie do hĺbky. Tieto algoritmy sa označujú ako

neinformované hľadanie, lebo nepoužívajú pri hľadaní žiadne dodatočné informácie. Prehľadávanie do šírky, ktoré najprv načíta stránky na jednej úrovni a potom ide na ďalšiu úroveň, je vo webových pavúkoch najpoužívanejšou technikou.

Druhá kategória grafových algoritmov je informované hľadanie. O každom prehľadávanom uzle existuje dodatočná informácia, ktorá sa môže použiť ako heuristika pri riadení prehľadávania grafu. Často používaným príkladom informovaného hľadania je algoritmus *najlepší najprv*. V tomto algoritme existuje heuristická funkcia *f(n)*, ktorá vo všeobecnosti môže závisieť od uzla *n*, cieľa hľadania, informácií, ktoré sa zozbierali po tento bod hľadania a ľubovoľných dodatočných znalostí o probléme. Najlepší najprv funguje ako rozšírenie prehľadávania do šírky, kde sa vždy expanduje ten uzol, ktorý má najvyššie ohodnotenie pomocou heuristickej funkcie. Typická implementácia algoritmu používa prioritný front. Vo webových pavúkoch sa pre heuristiky informovaného hľadania používajú rôzne metriky, napríklad počet vstupných odkazov, PageRank skóre, početnosť kľúčových slov, podobnosť s hľadaným dotazom.

Ďalšou kategóriou je paralelné hľadanie. Tieto algoritmy sa snažia spracovať rôzne časti vyhľadávacieho priestoru paralelne. Autori článku uvádzajú ako príklad pre paralelné hľadanie paralelnú aktivitu v umelých neurónových sieťach a genetické algoritmy, ale tieto dva príklady sú dosť vzdialené od klasických grafových algoritmov. Taktiež autori článku uvádzajú, že napriek veľkým možnostiam týchto algoritmov a použitiu v tradičných aplikáciách pre získavanie informácií (Chen, 1995), sa tieto algoritmy nepresadili vo webových aplikáciách.

3.1.3 Webové pavúky pre osobné hľadanie

Veľa webových pavúkov vyvinuli na pomoc jednotlivým používateľom pri vyhľadávaní užitočných informácií na webe. Keďže väčšinou tieto pavúky sa vykonávajú na klientských počítačoch, majú k dispozícii pre vyhľadávací proces väčší výpočtový výkon a pamäť. Taktiež tieto nástroje poskytujú používateľovi väčšie možnosti pri riadení a personalizácii vyhľadávacieho procesu.

Najznámejším skorým príkladom osobného webového pavúka je program tueMosaic (De Bra & Post, 1994). V programe tueMosaic používateľ zadal kľúčové slová, šírku a hĺbku hľadania a spustil pavúka, aby začal sťahovať stránky zo zadanej štartovacej stránky. Program používa „fish search“ algoritmus, čo je pozmenený „najlepší najprv“ vyhľadávací algoritmus.

Neskôr vývoj pavúkov prebiehal rôznymi smermi. Napríklad TkWWW robot bol vyhľadávač integrovaný do prehliadača TkWWW, pavúk SPHINX (Miller & Bharat, 1998) robil prehľadávanie do šírky a zobrazoval výsledok ako dvojrozmerný graf, CI Spider vykonával jazykovú analýzu a zhlukovanie nájdených výsledkov.

V iných štúdiách používajú pavúky v procese hľadania pokročilejšie algoritmy, napríklad Itsy Bitsy Spider (Chen et al., 1998) hľadá pomocou najlepší najprv prehľadávania a genetického algoritmu. Každé URL sa modeluje ako jedinec počiatočnej populácie. Kríženie sa definuje ako výber URL, na ktoré ukazujú začiatky viacerých URL. Mutácia sa modeluje pomocou načítania náhodného URL z adresára Yahoo. Pretože genetický algoritmus je optimalizačný proces, je vhodný pre hľadanie stránok na webe podľa určitých kritérií. V ďalších pavúkoch sa používalo simulované žíhanie, naivný bayesovský klasifikátor a ďalšie algoritmy.

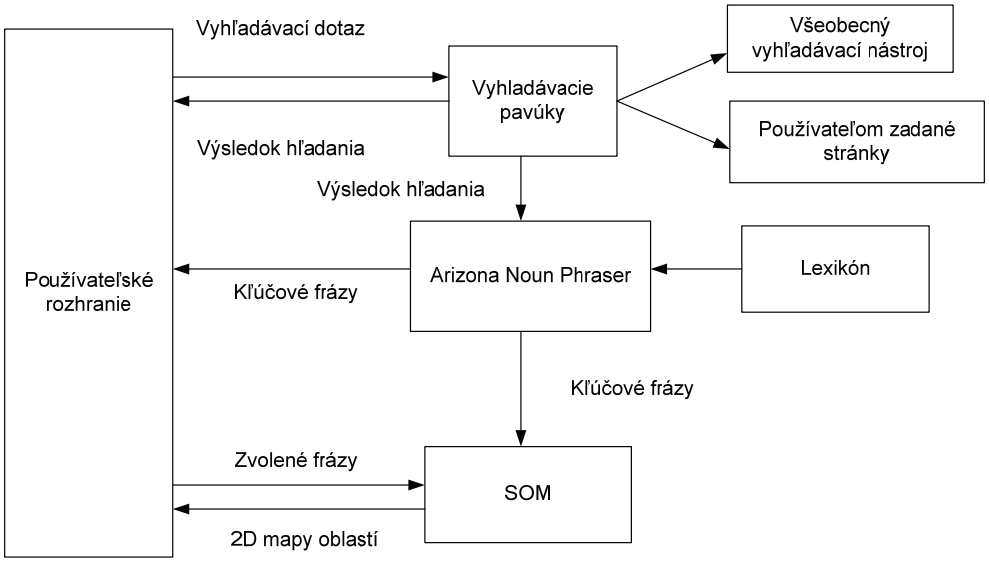
Osobitnou kategóriou pavúkov sú metapavúky. Sú to programy, ktoré sa pripájajú na iné vyhľadávacie stroje a sťahujú z nich výsledky. Kombinovaný výsledok poskytuje veľakrát pokrytie danej oblasti. MetaCrawler bol prvým metapavúkom. Poskytuje jednotné rozhranie k výsledkom z vyhľadávacích strojov Google, Yahoo! Search, MSN Search, Ask Jeeves, About, MIVA, LookSmart a ďalších. Meta-pavúk Dogpile poskytuje vyhľadávacie služby pre web, Usenet, žlté stránky, obrázky.

Nedávno vyvinuli vyhľadávacie pavúky pre peer-to-peer technológie. Napríklad JXTA Search (Waterhouse et al., 2002) používa ako svoj základ sieť Gnutella.

Prípadová štúdia

V tejto časti opíšeme architektúru dvoch vyhľadávacích agentov rozšírených o dodatočnú analýzu stiahnutých stránok. Prvý agent Competitive Intelligence Spider (skrátene CI Spider) je vyhľadávací agent (Chen et al., 2002), ktorý zbiera stránky v reálnom čase z používateľom špecifikovaných webových serverov a vykonáva nad nimi indexovanie a kategorizáciu tak, aby poskytol celkový prehľad daného webového servera.

Druhý nástroj je Meta Spider (Chen et al., 2001). Ten funguje podobne ako CI Spider, ale namiesto prehľadávania konkrétnej stránky do šírky sa pripája na vyhľadávací stroj na internete a integruje takto získané výsledky.



Obrázok 3-1. Architektúra agentov CI Spider a Meta Spider.

Na obrázku 3-1 je zobrazená spoločná architektúra pre CI Spider a Meta Spider. Štyri hlavné časti sú používateľské rozhranie, internetový pavúk, Arizona noun phaser, samorganizujúca mapa (SOM). Tieto časti spolupracujú pri hľadaní na webe a na analýze. Arizona noun phaser (Tolle & Chen, 2000) je indexovací nástroj pre indexovanie kľúčových fráz, ktoré sa nachádzajú v každom dokumente zozbieranom z webu pomocou internetových pavúkov. Nástroj vyberá z dokumentov na základe jazykového značkovania a jazykových pravidiel frázy obsahujúce podstatné mená.

SOM používa umelú neurónovú sieť pre automatické zhlukovanie webových stránok do rôznych oblastí dvojrozmernej mapy. Každý dokument sa reprezentuje pomocou vstupného vektora kľúčových slov a následne sa vytvorí výstupná dvojrozmerná mriežka. Po natrénovaní siete sa do nej posielajú dokumenty, ktoré sa rozdelia do zhlukov. Ku každej oblasti sa pridelí pomenovanie, čo je výraz, ktorý najpresnejšie reprezentuje zhluk dokumentov v danej oblasti. Dôležitejšie koncepty pokrývajú väčšiu oblasť a podobné koncepty sa zoskupujú blízko seba. Výsledná mapa sa zobrazí v používateľskom rozhraní, pričom si používateľ môže pozrieť dokumenty v jednotlivých oblastiach.

Pre porovnanie vlastností agentov CI Spider a Meta Spider sa vykonali dva experimenty, ktorých sa zúčastnilo 30 osôb. V prvom experimente porovnávali CI Spider so službou Lycos a manuálnym vyhľadávaním na serveri. Výsledky experimentu ukázali, že aj presnosť aj množstvo vracaných stránok pre CI Spider boli podstatne lepšie, ako v prípade služby Lycos pri 5% úrovni dôležitosti.

V druhom experimente porovnávali agent Meta Spider so službami MetaCrawler a NorthernLight. Čo sa týka presnosti, Meta Spider dosahoval podstatne lepšie výsledky ako každý z nich a najmä v porovnaní so službou NorthernLight. Množstvo stránok, ktoré vracal agent Meta Spider bolo porovnateľné so službou MetaCrawler a bolo lepšie, ako v prípade NorthernLight.

Hlavný dôvod pre lepšiu presnosť agentov bola ich schopnosť stiahnuť a skontrolovať obsah každej stránky v reálnom čase. To zabezpečuje, že každá zobrazená stránka obsahuje hľadané slová. Služby ako Lycos a NorthernLight pracujú pomocou indexov a tie môžu byť zastaralé. Vysoké množstvo vracaných výsledkov pre CI Spider sa dá zdôvodniť jeho hromadným sťahovaním stránok.

3.2 Reprezentácia, zdieľanie a získavanie znalostí na webe

Web v súčasnom „vývojovom štádiu“ charakterizuje množstvo obsiahnutých informácií, ale súčasne ich slabý formálny opis, čím sa podstatne zmenšuje množina znalostí, ktorú tieto informácie vytvoria. Navyše tieto znalosti sú zapísané rôznymi spôsobmi v rôznych jazykoch bez použitia ontológií. Ontológie sú veľmi variabilné a tým sa znižuje možnosť ich použitia.

Tento článok rozoberá možnú problematiku reprezentácie, zdieľania a získavania znalostí prostredníctvom webu so sémantikou. Zameriava sa najmä na prehľad rôznych pravidiel a vlastností, ktoré by mal web so sémantikou dodržiavať, aby ho bolo možné reálne zaviesť do praxe.

3.2.1 Web so sémantikou

Jedným zo základných kameňov vytvorenia webu so sémantikou je konceptualizácia dát dostupných na internete. Kľúčovým nástrojom na konceptualizáciu sú ontológie. Ontológie sa dajú charakterizovať ako formalizované reprezentácie znalostí určené k ich spoločnému používaniu a znovu použitiu. Ontológie sú často doménového (odborového) zamerania a zvyknú sa konštruovať ako konceptuálne (pojmové) hierarchie alebo siete.

(Polo)automatizované spracovanie informácií z webu so sémantikou, ktoré potom nasleduje, sa môže realizovať pomocou softvérových agentov, čo sú do určitej miery autonómne inteligentné programové komponenty pohybujúce sa zvyčajne v distribuovanom prostredí schopné realizovať „pre toho, kto ich poveril“ požiadavky na vyhľadávanie informácií a pod.

Dôležitým predpokladom webu so sémantikou je štandardizovaný opis webových zdrojov. Ako zdroj sa v tejto súvislosti môže chápať čokoľvek, čo je obsiahnuté v rámci webu (textové dokumenty, obrázky, videosekvencie, zvukové súbory a pod.). Každý zdroj by bol vybavený rovnakými charakteristikami (reprezentujúcimi jeho vlastnosti, vzťah k okoliu a pod.), čo by umožnilo používateľom pracovať s internetom ako s databázou znalostí prostredníctvom dopytovacích jazykov (podobných napríklad SQL).

Aby bolo toto možné dosiahnuť, treba definovať medzivrstvu, ktorá by umožňovala opísať obsiahnuté informácie. Táto vrstva sa nazýva metadáta. Môžu to byť stručne definované (štruktúrované) dáta o dátach., ktoré zachycujú obsah, kontext a štruktúru opisovaných dát.

Technologickým základom webu so sémantikou by sa podľa organizácie W3C mal stať štandard RDF (*Resource Description Framework*). Podľa oficiálnej definície ide o obecný rámec pre opis, výmenu a znovu použitie metadát. Rámec RDF poskytuje jednoduchý model pre opis zdrojov, ktorý nie je závislý od konkrétnej implementácie.

Tento prístup zabezpečuje určitý stupeň štandardizácie. Poskytuje však dosť značnú voľnosť pri tvorbe vrstvy metadát. Metadáta odrážajú do veľkej miery rozdielne prístupy jednotlivých tvorcov. V kombinácii so stratovou ontológiu spôsobili, že sa tento prístup zatiaľ nepresadil a stále je v štádiu výskumu. Na základe toho v ďalšej časti menujeme a rozoberáme vlastnosti, ktoré by mal web so sémantikou zahŕňať, aby ho bolo možné zaviesť do praxe.

3.2.2 Požiadavky na životaschopný web so sémantikou

Požiadavky na štandardizáciu knižníc a ontologických primitív

RDF nie je primárne výrazový jazyk, aj keď je obohatený o sémantiku. Toto rozšírenie sa realizuje štandardnými RDF schémami v kombinácii s DAML + OIL. Napriek tomu týmito spôsobmi nie je možné zapísať jednoduchú vetu *„5 osôb spolu tancuje“*.

Nedostatok výrazových prostriedkov RDF a absencia štandardných ontologických primitív núti poskytovateľov informácií reprezentovať informácie nevhodným a často sa vylučujúcim spôsobom. Oba dôsledky preto znemožňujú využitie týchto informácií, ich spoločné používanie a znovupoužitie.

Tu sa vyskytuje priestor pre mnohé formálne jazyky, napríklad *Z*, ktoré prichádzajú s matematickým aparátom, pomocou ktorého je následne možné konštruovať bloky znalostí (množiny, relácie, funkcie, čísla, sekvencie a pod.).

Podobný matematický aparát treba štandardizovať v RDF schémach, čím by sa vyriešil problém ich použitia, spoločného používania a znovupoužitia.

Požiadavky pre vyjadrovaciu notáciu

Zvyčajná vlastnosť výrazových notácií je prílišná zložitosť na úkor efektívnosti. V súvislosti s webom so sémantikou je toto tvrdenie vhodnejšie poopraviť do tvaru: „pokiaľ sa zvolí príliš zložitá notácia, ktorá sa používa cez určité rozhrania, až toto samotné použitie sa dá označiť za neefektívne“.

Obmedzené modely a notácie typu RDF + RDFS + DAML + OIL a RDF/XML nie sú vhodné na použitie ako medzi-jazyk, pretože istým spôsobom obmedzujú použitie danej reprezentácie znalostí.

Riešením tohto problému by sa mohlo zdať rozšíriť možnosti rozhraní, prípadne implementácia rôznych techník pre konkrétne účely.

Požiadavky pre notáciu vyššej úrovne

Problémom automatického získavania znalostí je veľká množina spôsobov, akými je možné danú znalosť zaznamenať a opísať. Tieto spôsoby značenia znalostí navyše nie sú medzi sebou zlučiteľné. Tieto problémy sa najvýraznejšie prejavujú, pokiaľ sa na zaznamenanie znalostí používajú nízko úrovňové jazyky (napríklad KIF).

Najdôležitejšou požiadavkou pri týchto prístupoch je znovupoužiteľnosť jazykových konštrukcií a takto zapísaných znalostí. Preto sa ako najvhodnejšie ukazuje zadefinovanie jazyka vyššej úrovne .

Na ilustráciu uvádzame možnosť zaznačenia určitej informácie rôznymi prístupmi. Ich výhody a prípadne nevýhody ponechávam na posúdenie čitateľovi.

```
E:
     Ned sold (the same) 3 cars twice on the 21/1/2001.
(This sentence does not specify whether the cars have been sold
individually,2 by 2, or 3 by 3. This ambiguity is kept in the
representations).

CGLF:
     [Person: Ned]<-(agent)<-[Sell: {*}@2]-
     {
        <-(object)<-[Car: {*}@3 @certain];
        <-(time)<-[Date: #21/1/2001];       }
FCG:
     [3 cars, object of: (2 sells, agent: Ned, time: 21/1/2001)]
FE:
     3 cars are object of 2 sells with agent Ned and time
      21/1/2001.
KIF:
     (forAllN 3 ?c car (forAllN 2 ?s sell
     (and (agent ?s Ned) (object ?s ?c) (time ?s '21/1/2001))))
PL:
     ∃cars set(cars) ∧ size(cars,3) ∧ ∀c ∈ cars

     ∃sells set(sells) ∧ size(sells,2) ∧ ∀s ∈ sells

     agent(s,Ned) ∧ object(s,c) ∧ time(s,21/1/2001)
RDF:
     <kif:Set ID="cars"><size>3</size></kif:Set>
     <rdf:Description aboutEach="#cars">
     <rdf:type resource="Car"/>
     <object><rdf:Description>
     <kif:Set ID="sells"><size>2</size></kif:Set>
     <rdf:Description aboutEach="#sell">
     <agent resource="Ned"/> <time>21/1/2001</time>
     </rdf:Description>
     </rdf:Description></object>
     </rdf:Description>
```

```
KIF2:
     definition of the ,``forAllN`` quantifier:
     (defrelation forAllN (?num ?var ?type ?predicate) :=
     (exists ((?s set)) (and (size ?s ?num)
     (truth ^(forall (,?var) (<= (member ,?var ,?s)
     (and (,?type ,?var) ,?predicate)))))))
```

Príklad 3-1. Prístupy zápisu určitej informácie rôznymi prístupmi.

Požiadavky na lexikálne/štrukturálne/ontologické konvencie

Ďalej uvedené požiadavky môžeme označiť za príklady pravidiel, ktoré treba štandardne dodržiavať.

Lexikálne pravidlá:

- používanie jednotného čísla,
- používanie slová v základnom tvare.

Štrukturálne pravidlá:

- tvorenie jednoduchých viet a vetných konštrukcií všade kde to je možné,
- špecializácia nových kategórií z existujúcich ontológií,
- používanie základných ontologických primitív.

Sémantické pravidlá:

- precíznosť,
- znovu používanie existujúcich znalostí.

Požiadavky na možnosť flexibilných odkazov na kategórie

Táto požiadavka nadväzuje na predchádzajúce pravidlá (najmä na znovupoužiteľnosť) a to z hľadiska odkazovania sa iných, existujúcich znalostí. Existuje mnoho prístupov, vzhľadom od použitej notácie. Najdôležitejšiu črtu, ktorú musia spĺňať všetky prístupy, je možnosť presnej referencie konkrétnej položky bez ohľadu na to kto ju vytvoril a do akej kategórie táto znalosť spadá. V tom prípade to znamená umožnenie odkazovania sa príslušnej znalosti v presnej skupine vytvorenej konkrétnym používateľom.

Požiadavky na centralizáciu

Najväčším nebezpečenstvom z hľadiska použitia previazaných znalostí a odkazovania sa je možnosť výskytu nekonzistentného stavu, t.j. stavu, v ktorom sa vzájomne odkazované znalosti medzi sebou odkazovaním vylučujú.

Na riešenie tohto problému treba potom nasadiť ontologické zhlukovacie techniky. Sú to rôzne semiautomatické algoritmy, kombinované s rôznymi heuristickými prístupmi. Zameriavajú sa nielen na riešenie vyskytnuvších sa problémov, ale najmä na ich predchádzanie.

3.2.3 Mechanizmy na zdieľanie a správu znalostí

Vzhľadom na možnosti a rozsah internetu treba, aby existovali určité pravidlá, ktoré by upravovali pridávanie, editovanie, mazanie znalostí a taktiež ich preväzovanie. Tieto pravidlá ostávajú zväčša platné bez ohľadu na to, či ide o jeden „znalostný server“ alebo či ide o viacero kooperujúcich serverov.

Základné pravidlá možno opísať takto:

1. Používateľ môže vymazať kategóriu, odkaz alebo graf (znalosti sa môžu reprezentovať grafom – pozri 3-2) iba ak ich vytvoril a táto operácia nespôsobí nekonzistentný stav. Pokiaľ vymazanú časť používa iný používateľ, v skutočnosti nedôjde k vymazaniu tejto časti, ale dôjde iba k presunutiu vlastníckych práv na používateľa, ktorý danú časť použil.
2. Autor kategórie môže modifikovať odkaz pripojený do tejto kategórie. Modifikácia odkazu prebehne vytvorením alternatívneho odkazu, pokiaľ modifikácia tohto odkazu spôsobuje nekonzistentný stav, prípadne používateľ nemá dostatočné práva na modifikáciu.
3. Používateľ môže pridávať graf alebo odkaz, dokonca aj keď nie je autor odkazovanej kategórie. Toto pridanie samozrejme nesmie spôsobiť nekonzistentný stav alebo generovať nadbytočnú znalosť.
4. Pokiaľ dôjde k jednej z predchádzajúcich akcií používateľom, ktorý nie je autor, autora, ktorého sa akcia dotýka, by mali automaticky informovať. Taktiež by sa uvedená zmena mala zaznačiť do logov.

Pridávanie znalostí

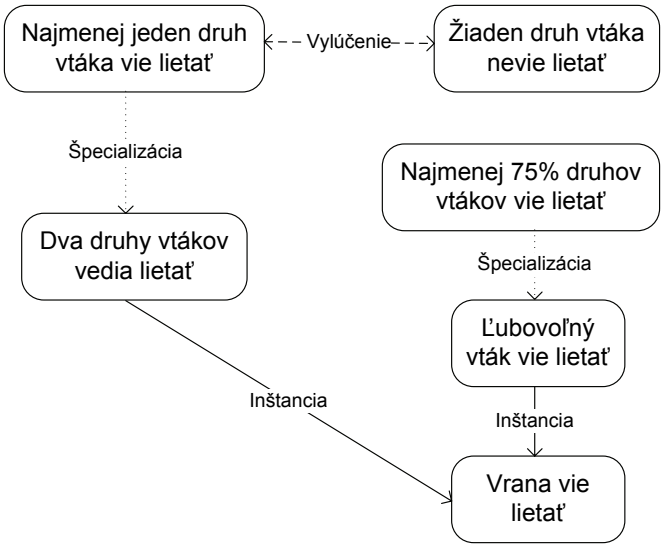
Vychádzajúc z predchádzajúcich poznatkov v budeme v ďalšom texte problematiku vytvárania a pridávania znalostí prezentovať na príklade (pozri obrázok 3-2).

- Na začiatku vytvárania znalostnej bázy môžeme predpokladať existenciu výroku „*Najmenej jeden druh vtáka vie lietať*“. Keďže tento výrok existuje ako prvý, možno ho automaticky pridať do bázy znalostí.
- Po tom môže vzniknúť požiadavka na pridanie výroku *„Najmenej 75% druhov vtákov vie lietať“*. Keďže oproti predchádzajúcemu – existujúcemu výroku nie je možné posúdiť jeho status, pridá sa ako ďalšia nezávislá znalosť.
- Status výrokov *„Ľubovoľný vták vie lietať“* a *„Dva druhy vtákov vedia lietať“* je ale na rozdiel od predchádzajúceho výroku možné určiť – predstavujú určité spresnenie predchádzajúcich výrokov – špecializujú ich.
- Výrok *„Vrana vie lietať.“* predstavuje konkrétnu inštanciu predchádzajúcich dvoch výrokov.
- Posledný výrok *„Žiaden druh vtáka nevie lietať.“* reprezentuje znalosť, ktorá sa vylučuje so znalosťami už zaradených v báze znalostí, preto na základe uvedených pravidiel nie je možné ju pridať a ďalej s ňou pracovať.

3.2.4 Zhodnotenie

V tejto časti sme predstavili základné črty a ciele webu so sémantikou. Opísali sme najvýznamnejšie problematické body, ktoré v súčasnosti znemožňujú využitie a priame nasadenie tejto technológie. Zároveň navrhujeme možné riešenia, ktorých zvládnutie by posunulo túto technológiu do roviny, v ktorej by sa mohlo zvažovať jej nasadenie a priemyselné využitie.

Celkové využitie tejto technológie po zvládnutí opisovaných problémov v rozsahu, v akom sa momentálne predpokladá, však ukáže čas.



Obrázok 3-2. Kontrola správnosti grafu reprezentujúceho znalosti.

3.3 Manažment znalostí na webe so sémantikou

Web výrazne zmenil dostupnosť elektronických informácií. Rastúci počet dokumentov na internete sťažuje vyhľadávanie, prístup, prezentáciu, správu informácií, ktoré požaduje široké spektrom používateľov. Systémy pre správu dokumentov majú dnes niekoľko slabých stránok:

1. vyhľadávanie informácií – existujúce vyhľadávanie podľa kľúčových slov môže vybrať nesúvisiace informácie, ktoré obsahujú kľúčové slová v inom význame, alebo sa nenájdu informácie, ktoré patria do hľadaného kontextu, ale neobsahujú kľúčové slová,
2. použitie informácií – dnes je potrebná práca človeka na to, aby sa z informačných zdrojov dali vytiahnuť len relevantné informácie a tie použiť ďalej,
3. spravovanie veľkých zle štruktúrovaných textových zdrojov je pracné a časovo náročné,
4. automatické generovanie dokumentov by malo umožniť adaptívnosť dynamických webových stránok.

Väčšina elektronických informácií je zle štruktúrovaných a na rôznych médiách. Firmy si zvykli na to, že ich intranety sa stali hodnotným úložiskom firemných znalostí. Problémom je správa týchto informácií, aby boli pre firmy použiteľnými znalosťami.

Tim Berners-Lee (Berners-Lee et al., 2002) predstavil víziu webu so sémantikou, ktorý poskytuje automatizovaný prístup k informáciám založený na strojovo spracovateľnej sémantike dát a metadát. Explicitná reprezentácia dát na základe doménových teórií (ontológií) posunie web do vyššej kvalitatívnej úrovne.

Ontológie (Fensel, 2001) sú kľúčovou technológiou pre web so sémantikou. Pojem vnikol (v tomto význame) v umelej inteligencii. V posledných rokoch sa stali populárnou výskumnou témou. Dôvod je vo vzájomnom a spoločnom pochopení informačnej domény medzi ľuďmi a strojmi.

Európsky projekt IST-1999-10132 *On-To-Knowledge* [4] obsahuje vyvinuté metódy a nástroje s ontologickým prístupom pre manažment znalostí. Tieto nástroje pomôžu pracovníkom firiem pristupovať k firemným informačným úložiskám efektívne, prirodzene, intuitívne.

3.3.1 Nástrojové prostredie pre manažment znalostí založený na ontológiách

Hlavným výstupom projektu *On-To-Knowledge* je sada softvérových nástrojov. Nástroje sú integrované do 3-vrstvovej architektúry:

1. horná používateľská vrstva
2. stredná vrstva (middleware)
3. spodná vrstva výberu informácií

Každý nástroj reprezentuje istú funkcionalitu. Celá sada nástrojov je postavená na základe DAML+OIL, RDF[5].

RDFferret: plnotextové vyhľadávanie – používateľský nástroj

Je to kombinácia plnotextového vyhľadávania a dopytovania cez RDF. Zdroje informácií s RDF metadátami sa dopĺňajú so zdrojmi informácií bez RDF metadát. *RDFferret* sa môže použiť ako bežný internetový vyhľadávač. Zadajú sa kľúčové slová alebo fráza, otázka v prirodzenom jazyku a vyhľadávač poskytne zoznam odkazov na relevantné webové stránky a to len vybraného typu. Doménové znalosti pre *RDFferret* sú vo forme ontológií špecifikovaných ako RDF schémy.

OntoShare: podpora komunít – používateľský nástroj

Umožňuje uloženie praktických informácií na základe ontológie a najpoužiteľnejšej informácie relevantnej pre spolupracovníkov. Ontológia pomáha novým spolupracovníkom orientovať sa vo svojej doméne a nazbieraných praktických skúsenostiach. Poskytuje štruktúru a jazyk pre spoločné používanie v skupine (komunite, pospolitosti). Každý používateľ (člen komunity) môže modifikovať časť ontológie.

Spectacle: prezentácia informácií – používateľský nástroj

Je to podklad pre prezentáciu obsahu, pomôcka pre používateľov systému. Prezentácia (štruktúrovanie, formátovanie, vykreslenie) je vhodne pripravená pre každého konkrétneho používateľa. Prezentuje sa obsah databáz, úložísk dokumentov a iných firemných informačných zdrojov a tiež sémantika informácií zo zdrojov webu so sémantikou. Nástroj pozostáva zo servera a programátorských knižníc pre generovanie prezentácií. Nástroj transformuje pre používateľa úlohu zbierania informácií získaných z vyhľadávania na úlohu prehliadania získaných informácií s pomocou heuristík. *Spectacle* poskytuje prezentácie v dvoch formách:

1. hypertextový výstup
2. grafická vizualizácia

---

[4] http://www.ontoknowledge.org

[5] http://www.daml.org, http://w3c.org

OntoEdit: vývoj ontológií – používateľský nástroj pre inžiniera znalostí

Je to prostredie pre vývoj v pracovnej skupine, ktoré je ľahko rozšíriteľné cez rámcový systém doplnkov. Tento grafický nástroj je nápomocný pre vývojárov ontológií v každom kroku práce. Umožňuje modelovanie nezávisle od konkrétnej reprezentácie jazyka. Poskytuje jednoduchý editor inštancií pre vkladanie faktov podľa modelovanej ontológie.

Konceptuálny model ontológie je uložený v internom formáte, ktorý môže byť prenesený do inej reprezentácie, napr. DAML+OIL, RDF, RDFS. Ontológie môžu byť použité nástrojom *Sesame*. Doplnky nástroja *OntoEdit*:

- OntoKick – príprava špecifikácie ontológie
- Mind2Onto – príprava poloformálnej štruktúry ontológie

Ontology Middleware Module – integrujúca platforma

Tento modul predstavuje administrátorské prostredie, aby sa rôzne nástroje pre manažmentu znalostí ľahšie integrovali do reálneho prostredia. Hlavné funkcie sú tieto:

- riadenie zmien, verzií pre ontológie (*OntoView*),
- riadenie prístupu, bezpečnosti,
- metainformácie pre ontológie, špecifické zdroje (triedy), tvrdenia (fakty).

Modul integruje ďalšie nástroje a technológie: *Sesame*, DAML+OIL, HTTP, RMI, SOAP.

OntoView: riadenie zmien v ontológiách – stredná vrstva

Vytvára a spravuje internú špecifikáciu vzťahov medzi rôznymi verziami jednej ontológie. Špecifikácia obsahuje metadáta o zmenách (autor, čas), konceptuálne vzťahy medzi verziami ontológie, transformácie medzi verziami ontológie. Porovnávacie funkcie rozlišujú tieto typy zmien:

- zmeny mimo logiky (v prirodzenom jazyku),
- zmeny v logickej definícii konceptu ontológie zasahujúce do formálnej sémantiky,
- zmeny názvoslovia (identifikátorov),
- pridanie definícií,
- odstránenie definícií.

Sesame: úložisko pre ontológie a pre dáta – stredná vrstva

Je to systém pre trvalé uchovanie RDF dát a schém a pre poskytovanie informácií za chodu. Systém *Sesame* je implementovaný v jazyku Java, aby bol prenositeľný na viaceré počítačové platformy. Prenositeľnosť systému pre ukladanie je umožnená pomocou štandardizovaného API – použiť možno akýkoľvek systém, ktorý ukladá RDF trojice, napr. DBMS. Podporovaný je jazyk RQL pre RDF dáta, RDF schémy, RDF grafy. *Sesame*[6] je vytvorené pod licenciou GNU LGPL.

[6] Sesame, http://sourceforge.net/projects/sesame/

CORPORUM: výber a úschova informácií – spodná vrstva

Sada nástrojov *CORPORUM* má dve hlavné úlohy – interpretácia textu v prirodzenom jazyku a vytiahnutie špecifických informácií z voľného textu. Druhá úloha si vyžaduje zásah človeka pre úspešnú činnosť.

CORPORUM pozostáva z dvoch častí:

- OntoExtract
- OntoWrapper

Vybrané informácie sa zapisujú v RDF(S)/DAML+OIL. Pridajú sa vhodné metadáta (*Dublin Core Meta Data*). Potom sa uložia do úložiska *Sesame*.

3.3.2 OIL: odvodená vrstva pre web so sémantikou

OIL (Fensel et al., 2000) je jazyk pre reprezentáciu ontológií. Základné požiadavky na jazyk OIL:

1. dostatočná vyjadriteľnosť pre aplikácie a úlohy spomenuté vyššie,
2. dostatočná formalizácia pre strojové spracovanie,
3. integrácia s existujúcimi webovými technológiami a štandardami.

Tento jazyk navrhli tak, aby kombinoval rámcové modelovanie, lepšiu vyjadriteľnosť, formalizmus a automatizované strojové spracovanie deskripčnej (opisnej) logiky. Jazyk OIL je vhodný aj pre web, pretože používa aj XML, aj RDFS, aj formalizovanú ľudsky čitateľnú formu. Rámcová štruktúra jazyka OIL je založená na XOL (*XML-based ontology exchange language*). Jazyk OIL obsahuje triedové výrazy (boolovské kombinácie tried), párové obmedzenia, definície tried (rámce), relačné vlastnosti, axiómy pre triedové výrazy.

Webové prostredie

RDF Schema je veľmi jednoduchý jazyk pre ontológie od W3C určený pre použitie vo webe so sémantikou. Tento jazyk poskytuje prostriedky na definovanie mien tried a vlastností, axiómy pre triedy a vlastnosti, doménové a rozsahové obmedzenia vlastností. Inštancie tried a vlastností sú definované v RDF. Jazyk OIL je nadmnožinou konštrukcií z *RDF Schema*.

Úrovne jazyka OIL

1. Core OIL (prienik Standard OIL a RDFS)
2. Standard OIL (neúplná nadmnožina RDFS)
3. Instance OIL (Standard OIL + inštancie)
4. Heavy OIL (Instance OIL + ďalšie budúce rozšírenia)

Jazyk OIL prijala spojená EU-USA iniciatíva, ktorá vyvinula jazyk DAML+OIL a štandardizovaný W3C. Ďalej W3C vyvíja jazyk OWL (*Ontology Web Language*).

3.3.3 Firemné aplikácie v sémantickom prístupe k informáciám

Vývoj nástrojov v projekte *On-To-Knowledge* bol silne motivovaný niekoľkými prípadovými štúdiami z firemného prostredia.

Metodológia projektu On-To-Knowledge

Východisko je v prípadových (uskutočniteľných) štúdiách – rozhodovanie choď/ nechoď, identifikácia používateľov, cieľová doména. Pokračuje sa v počiatočnom náčrte ontológie – špecifikácia požiadaviek, analýza zdrojov znalostí (vstupný slovník), vytvorenie poloformálnej definície ontológie. Ďalším krokom je čistenie definície ontológie – odhalenie znalostí doménovými expertmi, formalizácia (napr. OIL).

Cieľová ontológia sa posunie do fázy vyhodnocovania – kontrola požiadaviek, testovanie na cieľovej aplikácii, analýza vzorov použitia. Na základe ontológie sa vytvorí aplikácia, ontológia sa udržiava údržbou – zodpovednosť, ďalší vývoj.

Vyhľadávanie informácií

Jednu firemnú prípadovú štúdiu vytvorila spoločnosť Swiss Life. Problémom bolo vyhľadávanie relevantných častí textu vo veľmi veľkom dokumente o medzinárodných účtovných štandardoch na extranete. Ontológia vznikla automaticky naučením z dokumentu. Aplikácia pomohla používateľovi preformulovať otázku pomocou ponúknutých výrazov z ontológie. Ontológia bola štruktúrne jednoduchá a veľmi zlepšila výsledky vyhľadávania.

Manažment zručností

Druhá prípadová štúdia spoločnosti Swiss Life bola aplikácia na správu zručností, pracovných zaradení, dosiahnutého vzdelania zamestnancov. Aplikácia používala ručne konštruovanú ontológiu. Zaručená bola jednotnosť terminológie zadávaných údajov. Aplikácia umožňovala rýchle vyhľadávanie potrebných informácií.

Výmena znalostí vo virtuálnej organizácii

Prípadová štúdia, ktorú realizovala spoločnosť EnerSearch AB sa zamerala na vyhodnocovanie obchodnej hodnoty výsledkov projektu s prihliadnutím na potreby virtuálnej organizácie. Cieľom bolo zlepšenie transferu znalostí medzi rezidentnými zamestnancami a externistami cez existujúcu webovú stránku, tiež pomoc partnerom-akcionárom získavať aktuálne informácie o výskume a vývoji. Zameranie bolo na inteligentné plnotextové vyhľadávanie v textoch na webovej stránke. Použitý bol vyhľadávač *RDFferret*.

3.3.4 Zhodnotenie

WWW a firemné intranety zväčšili potenciál získavania a poskytovania elektronických znalostí. Projekt *On-To-Knowledge* je nevyhnutým krok v procese inovatívnych nástrojov pre spracovanie sémantiky.

Tento projekt a súvisiace nástroje sa zameriavajú na tri aspekty:

1. Získavanie, vytváranie ontológií a prepájanie ontológií s veľkým množstvom dát. Kvôli rozšíriteľnosti musí byť tento proces automatizovaný vo výbere informácií a spracovaní prirodzeného jazyka. Aby bola zaručená kvalita, je nutný vstup človeka pri zostrojovaní ontológií.
2. Ukladanie a správa ontológií a ich inštancií. Vyvinuli úložisko postavené na jazyku *RDF Schema*, ktoré poskytuje databázu.
3. Prehľadávanie sémanticky obohatených informačných zdrojov.

Vyvinutá technológia sa testovala na viacerých prípadových štúdiách. Je funkčná. Práca má aj pozitívne teoretické dôsledky. Počítače sú schopné spracúvať informácie

so strojovo spracovateľnou sémantikou. Sémantika prirodzeného jazyka sa dá transformovať do strojovo spracovateľného tvaru.

3.4 Ontológia – objavovanie taxonomických relácií z webu

Web vo svojej súčasnej forme zaznamenal významný úspech zvýšením počtu používateľov a zväčšením zdrojov informácií. Avšak, vzrastajúca komplexnosť webu sa neprenáša do súčasného stavu webových technológií. Ťažkosti s prístupom, extrahovaním, interpretáciou, udržovaním informácií zostávajú na používateľovi.

Tim Berners Lee – vynálezca webu – predstavil víziu webu so sémantikou. Web so sémantikou prinesie usporiadanie obsahu webových stránok, rozšíri súčasný web, v ktorom informácia bude mať presne definovaný zmysel. Web so sémantikou bude schopný podporovať automatické služby založené na sémantickom predpise. Tieto predpisy sa ukazujú byť kľúčovým faktorom v hľadaní východiska vzrastajúceho problému spájania rozširujúceho sa webového priestoru, kde väčšina zdrojov webu sa dá nájsť v súčasnosti iba cez kľúčové slová.

Ontológie sa ukázali byť správnou odpoveďou na problémy tým, že poskytujú formálnu konceptualizáciu osobitnej domény, o ktorú sa delí skupina ľudí. V kontexte webu so sémantikou ontológia opisuje teórie domén pre explicitnú reprezentáciu sémantiky dát. Web so sémantikou závisí od formálnej ontológie, ktorá vytvára štruktúru dátam, ktoré sú jednotné, prenosné a strojom pochopiteľné.

Napriek tomu, že nástroje ontologického inžinierstva sa za posledných desať rokov vyvinuli, ručná stavba ontológií zostáva únavnou, ťažkopádnou úlohou, ktorá môže ľahko vyústiť do získavania úzkeho profilu znalostí.

Úspech webu so sémantikou veľmi závisí od rozšírenia ontológie. V tejto časti sa budeme zaoberať základnou časťou ontologického inžinierstva, menovite rozvojom taxonomického základu ontológie. Cieľom je podať prieskum existujúcich prác na učenie taxonomických relácií z textov a príklad, aký môže mať tento prístup prínos.

Na začiatku uvedieme prieskumy existujúcich prác (Meadche & Staab, 2001, Staab et al., 2001, Sure et al., 2002). Budeme sa zaoberať symbolickým a štatisticky založeným prístupom. Rozdiel medzi nimi je, že štatisticky založený prístup umožňuje lepšie vyhodnotenie, meranie, ale symbolický prístup môže byť nakoniec precíznejší (Zhong et al., 2003). Cieľom práce je zosúladenie medzi týmito dvomi prístupmi a návrh nových algoritmov, ktoré budú detailne opísané a na príkladoch zhodnotené.

3.4.1 Prehľad symbolických prístupov

Myšlienku využívania lexikálno-symbolických vzorov vo forme regulárnych výrazov pre extrahovanie sémantických relácií v osobitých taxonomických reláciách predstavil Hearstom v (Hearst, 1992).

Prístupy založené na vzoroch sú vo všeobecnosti heuristickými metódami využívajúcimi regulárne výrazy, ktoré pôvodne úspešne aplikovali v oblasti extrahovania informácií v (Hoobs, 1993). V lexikálno-syntaktickej ontológii učebný prístup k textu vytvára predstavu pre prípady rozličných lexikálno-syntaktických vzorov, ktoré naznačujú reláciu záujmu napríklad taxonomickú reláciu. Hlavná myšlienka je

jednoduchá. Definovať regulárny výraz, ktorý zahŕňa znovuvyskytujúce sa výrazy a mapuje výsledky výrazov k sémantickej štruktúre.

Príklad: tieto príklady poskytujú vzorku vzorov, ktoré sú založené na scenári ontologickej extrakcie. Napríklad v (Hearst, 1992) jeden lexikálno-syntaktický vzor sa uvažuje v tvare:

*...NP { ,NP } * { , } or other NP ...*

Ak aplikujeme tento vzor na vetu, môžeme dospieť k záveru, že *NPs,* ktoré sa vzťahujú na základnú myšlienku naľavo od *or other* sú podmnožiny základnej myšlienky *NP,* ktorá je napravo od *or other.*

Napríklad veta:

Modriny, rany, zlomené kosti *alebo iné* zranenia sú bežné.

Extrahované taxonomické relácie sú: (modrina, zranenie), (rana, zranenie), (zlomená kosť, zranenie).

V (Hearst, 1992) sa vzory definovali manuálne, čo zaberalo veľa času a vznikalo veľa chýb. V (Morin, 1999) bola práca navrhnutá, podľa (Hearst, 1992) a rozšírená použitím symbolických nástrojov strojového učenia na zlepšenie lexikálno-syntaktických vzorov. V tejto základnej myšlienke prezentovali systém PROMETHEE, ktorý podporuje poloautomatické získavanie sémantických relácií a zlepšenie lexikálno-syntaktických vzorov.

Práca (Assadi, 1999) nám podáva správu o praktickom experimente konštrukcie regionálnej ontológie plánovania elektrických komunikačných sietí. Opísal zhlukovací prístup, ktorý kombinuje lingvistické a konceptuálne zásadné kritériá. Príkladom je vzor *NP,line,* ktorého výsledkom po modifikácii sú dve kategórie.

Faure a Nedellec v (Faure & Nedellec, 1998) prezentovali systém strojového učenia nazvaný ASSIUM, ktorý umožňuje získavanie taxonomických relácií zo syntaktických rozborov. ASSIUM sa zakladá na konceptuálnom zhlukovacom algoritme. Základné zhluky sú formované na základných slovách, ktoré sa vyskytujú s rovnakým slovesom a po rovnakej predložke. ASSIUM úspešne zhromažďuje zhluky na formovanie nových konceptov a hierarchie konceptov z ontológie.

Zlepšenie taxonomických relácií

Hahn a Schnattinger v (Hahn & Schnattinger, 1998) predstavili metodológiu pre podporu a zlepšenie doménovo špecifických taxonómií. Ontológia sa inkrementálne obnovuje ako sa získajú nové koncepty z reálnych textov. Prínos tohto procesu sa zameriava i na lingvistické a konceptuálne kvality rôznych foriem evidencie, ktorá tvorí základ produkcie a zlepšenia hypotéz konceptov. Osobitne zvažujú sémantické konflikty a analogické sémantické štruktúry zo základných znalostí do ontológie v snahe rozhodnúť o kvalite určitého návrhu. Takto rozšíria existujúcu ontológiu o koncepty a taxonomické relácie medzi konceptami.

System CAMILLE *Contextual Acquisition Machanism for Incremental Lexeme Learning* vynašli ako prirodzený jazyk, ktorý rozumie systému. Napríklad ak sa dostane na slovo, ktorému nerozumie, Camille sa snaží vyvodiť záver, aby sa dozvedel všetko o význame slova (Hastings, 1994). Ak je neznámym slovom podstatné meno, sémanticky limitujúce faktory na poskytnuté slovesá dávajú užitočné limity, ktoré sa týkajú významu podstatného mena. Učenie neznámych slovies je obtiažnejšie, teda

získavanie slovies bolo stredobodom pozornosti výskumu Camille. Camille testovali na niekoľkých skutočných svetových doménach známymi metódami získavania bodov za precíznosť a rozpamätanie. Rozpamätanie sa definovalo ako podiel v percentách správneho *hypobooku..Hypobook* považovali za správny, ak jeden z konceptov v *hypobooku* zamieril cieľový koncept. Presnosť je celkový počet správnych konceptov delený počtom konceptov vytvorených v celom *hypobooku.* Cammile dosiahla rozpamätanie 42% a prednosť 19% na sade 50 náhodne vybraných viet obsahujúcich 17 rôznych slovies.

3.4.2 Štatistický prístup

Postup na určovanie taxonomických relácií z textov zakladajúci sa na štatistike prezentovali viacerí autori. Medzi najznámejšie práce patrí práca (Kaufman & Rousseeuw, 1990), kde autori definovali zhlukovanie ako proces organizujúci objekty do skupín, v ktorých členy jednotlivých skupín sú si podobné. Definovali dva základné štýly zhlukovania:

1. nehierarchické zhlukovanie, v ktorej každý objekt je presne zaradený do jednej skupiny,
2. hierarchické zhlukovanie, v ktorej každá vhodná skupina sa skladá z menších podskupín. Na detailnú analýzu dát sa viac používajú hierarchické zhlukovacie algoritmy.

V roku 1993 napísali prácu (Periera et al., 1993). Autori vykonali zhlukovanie slov anglického slovníka.

V práci (Maanning & Schuetze, 1999) definovali dva základné prístupy hierarchického zhlukovania:

- hierarchické aglomeratívne zhlukovanie zdola nahor
- hierarchické divízne zhlukovanie zhora nadol.

Hierarchické aglomeratívne zhlukovanie zdola nahor

```
for i:= 1 to n do
  k_i:= x_i.
end for
K:= {k_1,...,k_n}.
j:= n+1.
while |K| > 1 do
  (k_n1,k_n2):=argmax_(ku,kv)∈K x K sim(k_u,k_v).
  kj = k_n1∪k_n2.
  K:= K\{k_n1,k_n2}∪{k_j}.
  j:= j+1.
end while
```

Príklad 3-2. Hierarchický zhlukovací algoritmus zdola nahor (Zhong et al., 2003).

Vstup: $X = \{x_1,\ldots,x_n\}$ je konečná n-prvková množina objektov.

Miera podobnosti medzi objektami: funkcia sim 2^x x $2^x \rightarrow R$

Výstup: hierarchický systém zhlukov

Postup:

- Zadefinuje sa miera podobnosti medzi objektami – každý prvok tvorí osobitný jednoprvkový zhluk.
- Pokiaľ nie je splnená podmienka (všetky objekty jeden zhluk) vyberú sa dva najpodobnejšie zhluky a zlúčia sa. Vytvorí sa nový rozklad Ω_i, kde i je zlučovacia hladina.

Týmto sa postupne vytvára podobnostný strom. Grafickým znázornením podobnostného stromu je dendrogram:

- Os *x* predstavuje množinu zhlukovacích hladín
- Os *y* poradové čísla zhlukovacích objektov

Hierarchické deliace zhlukovanie zhora nadol

```
K:= {X} (=k1)
j:= 1.
while ∃k_i ∈ K s.t.|k_i| > 1 do
  k_u:= argmin k_v∈K coh(k_v)
  (k_j+1, k_j+2) = split (k_u)
  K:= K\ {k_u}∪{ k_j+1, k_j+2}
  j:= j+2.
end while
```

Príklad 3-3. Hierarchický zhlukovací algoritmus zhora nadol (Zhong et al., 2003).

Z jedného zhluku sa vytvorí hierarchický systém zhlukov postupným delením zhlukov.

Vstup: $X = \{x_1,\ldots,x_n\}$ je konečná n prvková množina objektov

- Funkcia coh: $2^x \rightarrow R$ súvislosť
- Funkcia split: 2^x x $2^x \rightarrow 2^x$ rozdelenie

Výstup: hierarchický systém zhlukov

Postup:

- Zadefinuje sa miera podobnosti
- Načíta sa matica vzdialenosti všetkých dvojíc objektov
- Vypočíta sa priemerná vzdialenosť všetkých objektov od ostatných
- Lokalizuje sa objekt s najväčšou priemernou vzdialenosťou – tento je základom nového zhluku. Prvky sa rozdelia do dvoch zhlukov.
- Po skončení delenia jedného zhluku sa vyberie ďalší, ktorý sa delí na dva zhluky.
- Delenie končí, ak sú zhluky jednoprvkové

Podobnosť dokumentov

Kosínusová miera podobnosti, alebo normalizovaná korelácia koeficientov medzi dvomi vektormi x a y je daná vzťahom:

$$\cos(x, y) = \frac{\sum_{x \in X, y \in Y} xy}{\sqrt{\sum_{x \in X} x^2 \sum_{y \in Y} y^2}}$$

Nech sú dané funkcie p(x), q (x), pravdepodobnosť ich entrópie vypočítame:

$$D(p \,||\, q) = \sum_{x \in X} p(x) \log \frac{p(x)}{q(x)}$$

Príklad (Zhong et al., 2003):

Zisťovanie podobnosti medzi pojmami HOTEL a ACC.=ACCOMMODATION

ID	HOTEL	ACC	ADDRESS	WEEKEND	TENNIS
HOTEL	0	14	7	4	6
ACC	14	0	11	2	5
ADDRESS	7	11	0	10	3
WEEKEND	4	2	10	0	5
TENNIS	6	5	3	5	0

Pojem HOTEL je daný vektorom $x^T = (0,14,7,4,6)$

Pojem ACCOMMODATION je daný vektorom $y^T = (14,0,11,2,5)$

$$\cos(x, y) = \frac{(7*11 + 4*2 + 6*5)}{101*150} \approx 0{,}93$$

Pravdepodobnosť výskytu pojmu HOTEL je daná (0,0.45,0.22,0.13,0.19)

Pravdepodobnosť výskytu pojmu ACCOMMODATION je daná (0.44,0,0.34,0.06,0.16)

D(HOTEL || ACCOMMODATION) = 0.22*(0.22/0.34)+ ...+0.19*(0.19/0.16) ≈0.65

3.4.3 Experimenty

Často používanou metódou na hľadanie taxonomických relácií je metóda najbližších susedov (kNN) (Hearst & Schütze, 1993).

Metóda najbližších susedov:

- podľa zvolenej funkcie podobnosti vyhľadá k najbližších susedov z trénovacej množiny,
- na základe tried priradených najbližším susedom a podobnosti určí skóre dokumentu,
- ak skóre prekročí danú hranicu, dokument sa zaradí do príslušnej triedy.

Podobnou metódou je metóda založená na kategorizácii (Resnik, 1993). Ďalšími metódami, ktoré sa použili na experimenty sú centroidná metóda (Periera et al., 1993), stromová zostupná a stromová vzostupná metóda (Meadche & Staab, 2000).

Na experimenty použili ontológiu z projektu GETESS – 1999. Ontológia obsahovala 1052 slov a fráz, ktoré rozdelili do 182 tried. Dáta, ktoré použili na experiment boli webové stránky hotelov po celom svete – cca 988 000 slov.

Slová podľa frekvencie rozdelili do troch skupín: 0 – 40, 40 – 500 a viac ako 500.

Použili sa miery podobnosti:

- JC – Jacquard coeficient,
- L1 – L1 metrika,
- SD – Skew,
- Divergence.

Tabuľka 3-1. Metóda kNN, k=30[5] (Zhong et al., 2003).

	0-40	40-50	>500	Overall
JC	.33773/.17142	.33924/.15384	.40181/.12457	.37044/.15211
L1	.33503/.16428	.38424/.21025	.38987/.14471	.37636/.17195
SD	.31505/.14285	.36316/.18461	.45234/.17845	.38806/.17063

Tabuľka 3-2. Metóda založená na kategorizácii (Zhong et al., 2003).

	0-40	40-50	>500	Overall
JC	.26918/.12142	.34743/.17948	.47404/.28282	.37554/.2023
L1	.27533/.125	.41736/.25128	.56711/.38383	.43242/.26190
SD	.28589/.12857	.34932/.18461	.51306/.31649	.39755/.21957

Tabuľka 3-3. Centroidná metóda (Zhong et al., 2003).

	0-40	40-50	>500	Overall
JC	.17362/.07831	.18063/.08119	.30246/.14434	.22973/.10714
L1	.21711/.09793	.30955/.13938	.37411/.1687	.30723/.12698
SD	.22108/.09972	.23814/.11374	.36486/.16147	.28665/.10714

Tabuľka 3-4. Stromová zostupná metóda (Zhong et al., 2003).

	0-40	40-50	>500	Overall
JC	.00726/0	.01213/.00512	.02312/.0101	.014904/.005291
L1	.08221/.03214	.05697/.02051	.21305/.11111	.128844/.060846
SD	.08712/.03214	.07739/.03589	.16731/.06734	.011796/.047619

Tabuľka 3-5. Stromová vzostupná metóda + kNN, k=15 (Zhong et al., 2003).

	0-40	40-50	>500	Overall
JC	.32112/.075	.33553/.0923	.40968/.08754	.36643/.08597
L1	.33369/.07142	.34504/.0923	.42627/.09764	.38005/.08862
SD	.31809/.06785	.32489/.05128	.45529/.11111	.38048/.08201

Tabuľka 3-6. Stromová vzostupná metóda + kNN, k=30 (Zhong et al., 2003).

	0-40	40-50	>500	Overall
JC	.34444/.16428	.35858/.14358	.41260/.10774	.38215/.14021
L1	.35147/.16428	.36545/.15384	.41086/.11784	.38584/.14682
SD	.32613/.13571	.36485/.1641	.45732/.16498	.39456/.1574

3.4.4 Zhodnotenie

Cieľom tejto práce bolo:

- Rozvoj taxonomického základu ontológie. Taxonómia sa definuje ako systematické triedenie a opis javov určitej špecifickej oblasti.
- Prieskum existujúcich prác. Opis rôznych prístupov na riešenie problému – 1. symbolický prístup, 2. štatistický prístup.
- Experimenty a návrh nových kombinovaných algoritmov.

Výsledkom práce je zosúladenie analyzovaných postupov a návrh nových algoritmov pre rozvoj taxonómie zahŕňajúci existujúce taxonomické relácie ako pôvodné znalosti.

Najlepšie výsledky experimentov sa dosiahli pri kombinácii stromovej vzostupnej metódy s metódou kNN pri k=15 (Zhong et al., 2003).

4 INFRAŠTRUKTÚRA PRE WEBOVÉ INTELIGENTNÉ SYSTÉMY

Rýchlym rozšírením osobných počítačov za posledných pätnásť rokov sa radikálne zmenil svet nejedného z nás. To, čo kedysi bolo nepredstaviteľné, ako napríklad okamžitý prístup k terabajtom dát a ich prehľadávanie za niekoľko sekúnd, je dnes realitou. Máloktorý používateľ si však uvedomuje, že za tieto služby vďačí nielen „hmatateľnému" hardvéru, ale najme sofistikovaným algoritmom softvéru.

Problém minulosti s nedostatkom informácií, sa postupom času pretransformoval na problém vyhľadávania informácií medzi obrovským množstvom dát, ktoré máme v súčasnosti k dispozícii. Nájdené informácie veľmi často nie sú zaujímavé len pre jedného koncového používateľa, ale aj množstvo používateľov v jeho okolí (informácie o práci, zábave a podobne). Tieto informácie má preto vo väčšine prípadov zmysel „posunúť" bližšie k používateľom. Práve na to využívame jednu z najčastejšie využívaných „neviditeľných" služieb – službu predprípravy, a to na takmer každej úrovni využitia širokého spektra výpočtových prostriedkov. Algoritmy na vyhľadávanie a predprípravu patria medzi najvyužívanejšie algoritmy vo všeobecnosti, práve preto je vhodné im venovať aj primeranú pozornosť.

4.1 Algoritmické aspekty webových inteligentných systémov

Celosvetová pavučina sa stala v poslednej dekáde jednou z najvýznamnejších oblastí výskumu. Má mnoho spoločného s tradičnými oblasťami výskumu, no súčasne tu nachádzame mnoho odlišného a web otvára nové výzvy pre výskumníkov. Ako príklad môže slúžiť náhľad na web ako na veľkú distribuovanú databázu. Web sa však od tradičných databáz podstatne líši spôsobom, ako sa ukladajú informácie, veľkosťou, počtom používateľov, ktorý súčasne k webu pristupujú.

Táto práca sa zaoberá algoritmickými aspektami webových inteligentných systémov. Tieto aspekty sa opisujú na konkrétnom systéme, ktorým je online portál zábavy s názvom myFreddy. MyFreddy vystavali na konceptoch z viacerých oblastí výskumu: ide o strojové učenie a štatistiku, teóriu agentov a na myšlienkach efektívnej a rýchlej manipulácie s dátami a prítulného používateľského rozhrania.

MyFreddy vytvorili ako webovú pospolitosť založená na hlasovaní používateľov. Kľúčovými vlastnosťami systému sú:

- *podpora viacerých typov dát.* Podporuje sa je niekoľko typov dát (text, obrázky, zvuk, video) a jazykov;
- ľahká navigácia;
- *obsah sa čerpá výlučne od používateľov.* Používateľské rozhranie umožňuje používateľom pridávať do systému rôzny obsah. Na druhej strane používateľ môže monitorovať, ako obsah, ktorý pridal, ďalej prijímajú ostatní na základe ich hlasovania;
- *zábavný obsah.* Obsah sa zredukuje na taký, ktorý produkuje zábavu: anekdoty, komiksy, kreslené vtipy;
- jadrom systému je vizuálny webový agent;
- *rozšírená analýza dát.* Súčasťou portálu sú algoritmy narábania s dátami a analýzy zbieraných dát: výber nasledujúceho dokumentu na hlasovanie, odporúčanie dokumentov používateľom, predpovedanie hlasovania, združovanie príbuzných dokumentov.

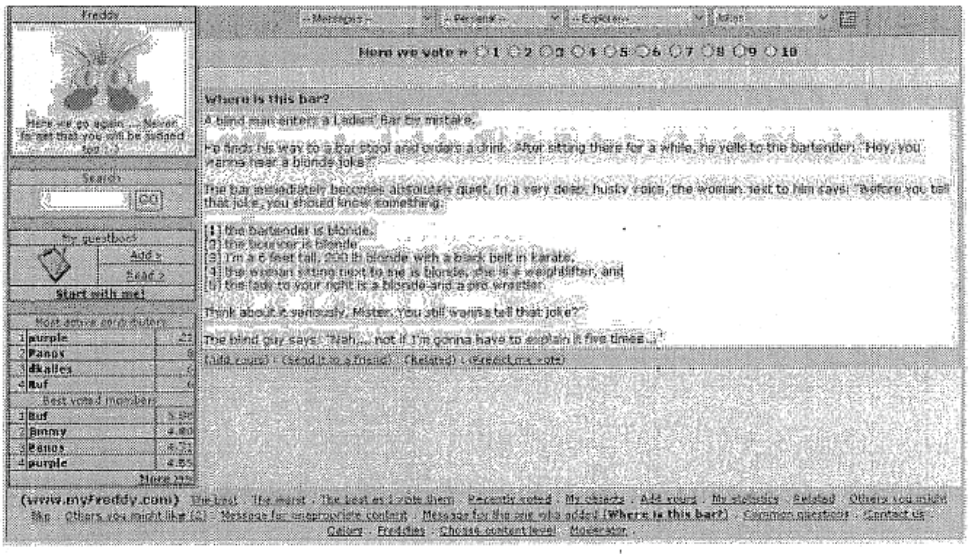
4.1.1 Stručný opis systému

Pokiaľ ide o používateľské rozhranie, v MyFreddy používateľ musí hlasovať pre každý ponúknutý objekt na to, aby mohol vidieť ďalší. Hlasovanie je povinné na to, aby si používateľ mohol prezerať obsah sídla. Má to výhodu v tom, že takto sa zozbiera dostatočné množstvo dát, ktoré charakterizujú používateľov a dajú sa ďalej použiť.

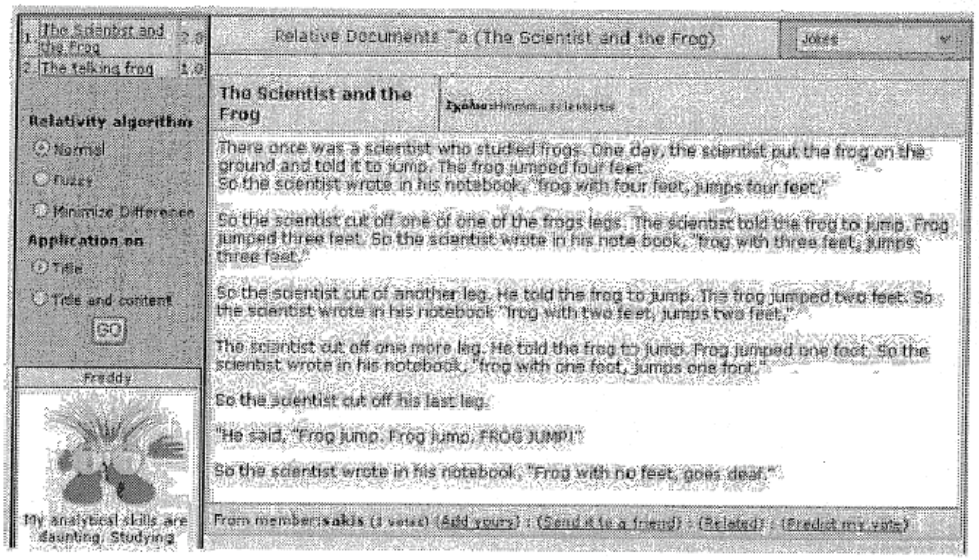
Dajme tomu, že po prihlásení sa do systému si používateľ zvolil voľbu *Jokes*. Používateľovi sa zobrazí stránka, na ktorej sa už ponúka jeden vybratý vtip, za ktorý môže hlasovať voľbou od 1 do 10 Na základe jeho hlasovania sa mu vyberie a zobrazí ďalší vtip, za ktorý môže opäť hlasovať. Na tejto stránke sú tiež odkazy, pomocou ktorých používateľ môže pridať nový vtip, poslať vtip priateľovi, nájsť príbuzné dokumenty, či predpovedať hlasovanie. Ak si používateľ zvolí voľbu *Related*, tak sa mu zobrazí stránka podobná tej na obrázku 4-2. Tu si používateľ môže vybrať jeden z troch algoritmov na načítanie rôznych typov príbuzných dokumentov.

Obrázok 4-3 predstavuje používateľovu osobnú stránku, na ktorej môže monitorovať hodnotenie ním zadaných napr. vtipov. Je tu napríklad vidieť počet jednotlivých hlasov pre každú známku od 1 po 10 a tiež priemerná známka, či celkový počet hlasov, ktoré vtip dostal.

Pokiaľ ide o výkonnosť, opisovaná aplikácia je typu zákazník-obsluha, čo znamená veľkú záťaž servera, predovšetkým pre charakter implementovaných algoritmov. V tomto smere môže značne pomôcť *caching*. Netýka sa hlavnej pamäti, pretože programová inštancia je v pamäti len počas konštrukcie tej-ktorej stránky, po jej vytvorení je alokovaná pamäť uvoľnená. *Caching* sa týka databázy; dáta sa do cache pamäti uchovávajú počas toho, čo si používateľ prehliada rôzne stránky. Ako príklad môže slúžiť množina odporúčaných dokumentov - sú vypočítané raz, keď o to používateľ požiada a zostávajú v *cache* pamäti a používateľ môže cez ne prechádzať bez plytvania zdrojmi.



Obrázok 4-1. Hlavná stránka hlasovania pre vtipy.

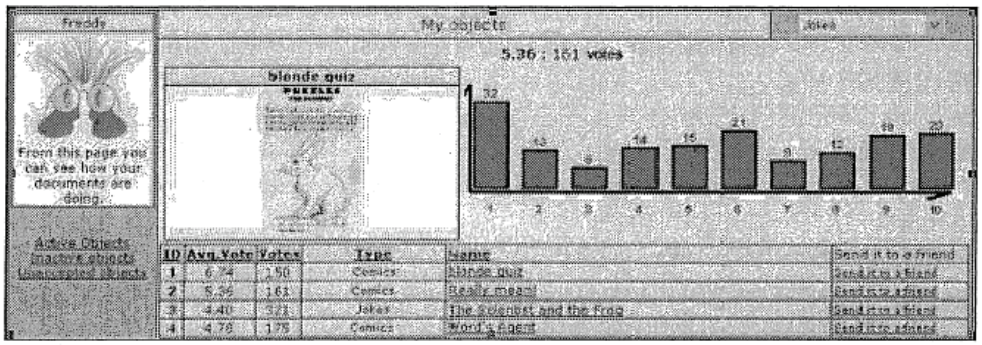


Obrázok 4-2. Odporučenie príbuzného dokumentu použitím rôznych prístupov (zdroj: Zhong et al., 2003).

Pokiaľ ide o autentifikáciu používateľov, sú dva typy používateľov. Prvá skupina sa hlási do systému cez login/password systém. Je to výhodné pre používateľov, ktorí pristupujú do systému nie iba raz, ale viackrát, napr. pre pridávanie objektov (vtipov, obrázkov,...) do systému. Login a password však môže byť príťažou pre používateľa, ktorý chce len prechádzať cez obsah.

Pre takýchto používateľov sa prijalo riešenie cez *cookies*. Keď používateľ prvýkrát vstúpi do systému, vyhľadá sa špecifický identifikátor v jeho počítači a ten sa odovzdá aktuálnemu sedeniu prehliadača a server ho použije na označenie jeho hlasovania

a iných akcií. Keby sa ten špecifický identifikátor nenašiel, tak sa jeden náhodne vytvorí a odovzdá jeho počítaču pre ďalšie použitie. Opísaný prístup má jednu zrejmú nevýhodu: používateľa možno identifikovať, ak pristúpi do systému len z jeho špecifikovaného prostredia.



Obrázok 4-3. Osobná stránka používateľa a detaily ním zadaných objektov (zdroj: Zhong et al., 2003).

Freddy je vizuálny webový agent. Reaguje na používateľove vstupy tým, že vyjadruje komentáre alebo poskytuje nápomocné tipy. Používa hlasovanie používateľa, jeho relatívnu pozíciu v systéme a charakteristiku hodnoteného dokumentu na vygenerovanie dynamickej odpovede. Plánuje sa rozšíriť Freddyho vstupy o tieto dáta: priemerný čas medzi hlasovaniami, čas, ktorý uplynul od poslednej návštevy systému, charakteristiky postupnosti hlasovaní, denná doba.

4.1.2 Algoritmy

Webová pospolitosť MyFreddy zbiera veľké množstvo dát, ktoré sa musia nejakým spôsobom analyzovať. Potenciálnymi cieľmi analýzy dát sú:

- ponúknuť objekty príbuzné aktuálne prezentovanému,
- vybrať ďalšie objekty, ktoré by sa používateľovi mohli páčiť,
- predpovedať používateľovo hlasovanie,
- odporučiť členov komunity s podobnými preferenciami.

Výber nasledujúceho dokumentu

Keď používateľ hlasuje, ihneď sa mu je predloží nový dokument. Keby sa tento dokument vyberal úplne náhodne, malo by to isté nevýhody: používateľ by mohol hlasovať niekoľkokrát za sebou pre ten istý objekt. Do úvahy pripadali dve základné techniky:

- *Náhodný výber z objektov, pre ktoré používateľ ešte nehlasoval.* Dá sa efektívne vypočítať pomocou štandardných indexovacích techník v databáze. Špeciálny prípad nastáva, keď používateľ videl už všetky objekty nejakého typu. Vtedy mu algoritmus vráti objekt, ktorý videl najskôr.
- *Náhodný výber z objektov, ktoré sa mu pravdepodobne budú páčiť.* Táto technika odfiltruje objekty nielen podľa toho, či pre ne používateľ hlasoval alebo nie, ale aj na základe odhadu, ako zaujímavý by sa objekt používateľovi videl. Algoritmus

využíva predpovedanie hlasovania, ktorému sa budeme venovať v ďalších častiach. Hlavnou myšlienkou je odfiltrovať všetky náhodne vybrané objekty, pre ktoré predpovedané hlasovanie je menšie ako istá hranica. Táto hranica je jedným zo vstupov algoritmu a pomocou nej možno nastaviť ako časovo (a v konečnom dôsledku aj pamäťovo) náročný bude algoritmus. Ak bude táto hodnota vysoká, tak v priemere bude algoritmu trvať dlhšie, kým nájde vhodného kandidáta. Tento účinok sa môže zredukovať zavedením hornej hranice pre počet objektov, ktoré sa môžu skontrolovať. Keď sa táto hranica prekročí, algoritmus vráti dovtedy najlepšieho kandidáta.

Pseudokód algoritmu, ktorý vracia objekty, ktoré by sa používateľovi mohli páčiť:

```
Vstup. Používateľ U, kolekcia dokumentov S, hranica hlasovania
V1, hranica počtu kontrolovaných dokumentov C1.
Výstup. Dokument, pre ktorý používateľ ešte nehlasoval a ktorý
by sa mu mohol páčiť alebo dokument, ktorý videl najskôr
(v prípade, že už hlasoval pre všetky dokumenty).
Algoritmus.
Načítaj množinu Sv objektov, pre ktoré používateľ ešte
nehlasoval.
ak |S| > |Sv| tak
{    Náhodne vyber objekt X z množiny S - Sv.
     Pre používateľa U predpovedaj hlasovanie V pre object X.
     ak V < V1 tak
     {      Ulož X a jeho hodnotu V.
            Zvýš hodnotu počítadla C o jednotku.
            ak C = C1 tak vráť objekt X s najvyššou hodnotou
                  V spomedzi C1 uložených objektov
            inak vráť sa na náhodný výber ďalšieho objektu.
     }
     inak vráť objekt X.
}
ak |S| = |Sv| tak
{    vyber Sx ako množinu |S|-1 posledných objektov, pre
     ktoré používateľ hlasoval.
     Vráť jediný object patriaci do S - Sx.
}
```

Nájdenie zaujímavých objektov a predpovedanie hlasovania porovnávaním používateľov

Hlavnou myšlienkou je ku používateľovi A nájsť používateľa B, ktorý je k nemu najbližšie, pričom vzdialenosť používateľov $D(A, B)$ *sa* definuje takto: nech A_d, resp. B_d je množina dokumentov, pre ktoré hlasoval používateľ A, resp. B a nech $C(A, B) = A_d \cap B_d = \{Z_1, Z_2, ...\}$ je množina dokumentov Z_i, pre ktoré hlasovali obaja používatelia A aj B. Nech $V_{X,Y}$ označuje hlasovanie používateľa X pre dokument Y. Potom vzdialenosť dvoch používateľov sa dá vypočítať pomocou funkcie zohľadňujúcej rozdielnosť hlasovania pre dokumenty, pre ktoré hlasovali obaja používatelia:

$$D(A,B) = \sqrt{\sum_{i=0}^{|C(A,B)|} (V_{A,Z_i} - V_{B,Z_i})^2} \ .$$

Ak používatelia A a B sú si blízki, tak budú (s istou pravdepodobnosťou) podobne hlasovať aj pre dokumenty, pre ktoré B hlasoval, ale A nie. Teda, ak vrátime dokumenty, pre ktoré B hlasoval kladne a A nehlasoval, tak s veľkou pravdepodobnosťou ich kladne prijme aj používateľ A. Nefunguje to celkom, ak používateľ B hlasoval za menej dokumentov než používateľ A, alebo ak prevyšujúce dokumenty nehodnotil dobre. Vtedy nemôžeme vybrať dokument zaujímavý pre A. Rozšírením tejto myšlienky je počítať k najbližších susedov používateľa A.

Pseudokód algoritmu, ktorý vracia zaujímavé dokumenty vyhodnotením niekoľkých používateľov blízkych používateľovi A je:

Vstup.
Používateľ A, maximálny počet používateľov x na porovnávanie, maximálna veľkosť množiny dokumentov Δ, maximálny počet dokumentov t, ktoré sa vrátia, maximálny počet používateľov z, na základe ktorých budeme vyberať vrátené objekty.

Výstup.
Množina dokumentov, pre ktoré používateľ A nehlasoval a pravdepodobne sa mu budú páčiť.

Algoritmus.
1. Vytvor množinu U_x prvých x používateľov s najvyšším počtom hlasovaní
2. Pre každého používateľa $B \in U_x$ vytvor množinu $C_\Delta(A,B)$ ($|C_\Delta(A,B)| \leq \Delta$) objektov, pre ktoré hlasovali A aj B.
3. Vypočítaj $D(A, B)$ pomocou množiny $C_\Delta(A,B)$
4. Vlož používateľa B do zoznamu používateľov usporiadaného podľa rastúcej vzdialenosti od používateľa A.
5. Vytvor množinu používateľov K vybratím prvých z používateľov z usporiadaného zoznamu používateľov.
6. Pre každého používateľa z K vyber t najlepšie ohodnotených dokumentov, pre ktoré používateľ hlasoval, ale A nehlasoval.
7. Vráť zjednotenú množinu t najlepšie ohodnotených dokumentov vybraných na základe priemerného hlasovania všetkých používateľov z K.

Mierne modifikovaný prístup sa dá použiť aj pre predpovedanie hlasovania pre konkrétny objekt používateľom A: nájde sa používateľ B najbližší k A, ktorý hlasoval pre daný objekt. Môžeme potom predpokladať s istou pravdepodobnosťou, že používateľ A bude pre daný objekt hlasovať rovnako, ako používateľ B.

Tento prístup má nevýhodu v tom, že je časovo náročný. Na dobrú predpoveď musíme porovnávať používateľa A s veľkým množstvom potenciálnych používateľov B. Pre rýchlu odozvu musíme potom výrazne zredukovať množinu používateľov, s ktorými A porovnávame, čím trpí presnosť predpovede. Výhodou je, že tento prístup sa dá použiť nielen pre textové dokumenty.

Nájdenie zaujímavých objektov a predpovedanie hlasovania použitím naivnej Bayesovej analýzy

V tomto prístupe sa pre jedného používateľa A analyzujú minulé hlasovania pre dokumenty v závislosti od obsahu týchto dokumentov. Dá sa použiť len pre textové dokumenty. Vo všeobecnosti máme funkciu f: $X \rightarrow V$, ktorej hodnoty máme predpovedať. V našom prípade X je množina dokumentov a V sú možné hodnotenia

týchto dokumentov. Dokumenty x z množiny X sa reprezentujú m-ticami $\langle a_1, a_2, ..., a_m \rangle$, ktoré sú vlastne množinou všetkých slov v dokumente. Úlohou je vypočítať čo najlepšiu predpoveď *Pred* pre hodnotu $f(x)$ na základe nasledujúcich pravdepodobností:

$$\Pr ed = \max_{u_j \in V} \Pr(u_j \mid a_1, a_2 ... a_m) = \max_{u_j \in V} \Pr(a_1, a_2 ... a_m \mid u_j) \Pr(u_j)$$

Naivný Bayesov klasifikátor vychádza z predpokladu, že atribúty sú nezávislé

$$\Pr(a_1, a_2 ... a_m \mid u_j) = \prod_i \Pr(a_i \mid u_j)$$

z čoho dosadením získame

$$\Pr ed = \max_{u_j \in V} \Pr(u_j) \prod_i \Pr(a_i \mid u_j)$$

Potrebné pravdepodobnosti $\Pr(u_j)$ a $\Pr(a_i|u_j)$ môžu byť vypočítané nasledovne

$$\Pr(u_j) = \frac{|docs_j|}{|docs|} \qquad \Pr(a_i \mid u_j) = \frac{n_i + 1}{n + |vocabulary|}$$

kde $docs_j$ je množina dokumentov, ktoré používateľ ohodnotil známkou j, *docs* je množina všetkých dokumentov, ktoré používateľ hodnotil, n je celkový počet slov v kolekcii $docs_j$, n_i je počet výskytov slova a_i v kolekcii $docs_j$, a *vocabulary* je kolekcia všetkých rôznych slov v *docs*.

Proces vytvorenia potrebných štruktúr pre naivnú Bayesovu analýzu vyhodnotením textového obsahu každého objektu, pre ktorý používateľ hlasoval, nazývame procesom trénovania. Jeho výstupom je model používateľa, ktorý charakterizuje jeho správanie pri hlasovaní. V druhom kroku vypočítame pravdepodobnosti, s ktorými nový dokument, pre ktorý používateľ ešte nehlasoval, bude patriť do jednotlivých klasifikačných stupňov (v našom prípade od 1 do 10). Najvyššia pravdepodobnosť určuje hľadaný klasifikačný stupeň – to bude naša predpoveď hlasovania.

Problémom pri tomto prístupe je výpočtová náročnosť: systém musí analyzovať veľké množstvo dokumentov. Toto sa dá trochu obísť inkrementálnymi obnovami (*update*ami) používateľovho modelu po každom jeho hlasovaní. Navyše po dostatočnom množstve trénovacích dokumentov štruktúry naivnej Bayesovej analýzy budú dostatočne robustné na generovanie presných predpovedí. Po tomto okamihu už nemusíme robiť obnovu modelu, čím šetríme zdroje.

Vyhľadávanie príbuzných dokumentov

Podstata procedúry, ktorú nazývame štandardným hľadaním, tkvie v tom, že vezmeme východiskový dokument a aplikujeme naň rozšírené hľadanie: pre každé slovo v dokumente rozbehneme vyhľadávanie tohto slova v ostatných dokumentoch a vrátime dokumenty s najväčšími počtami agregovaných výskytov týchto slov.

Nevýhodami štandardného hľadania sú: nedokáže rozpoznať relatívnu dôležitosť slov, napr. slovo blondína je veľmi dôležité pri určovaní príbuznosti vtipov. Okrem toho, veľa slov má malú informačnú hodnotu (spojka a). Takéto vyhľadávanie uprednostňuje veľké dokumenty.

Výhodou je, že sa dá efektívne implementovať pomocou štandardných databázových indexovacích a vyhľadávacích procedúr.

Niekedy sa uvažujú aj opatrenia proti zvýšenému šumu. Pod šumom rozumieme okrajové fakty, ktoré robia hľadanie menej efektívnym (napr. použitie veľkých a malých písmen). Na jeho odstránenie sa slová normalizujú, t.j. odstraňujú sa špeciálne znaky a slová sa konvertujú na malé písmená. Tiež sa filtrujú slová kratšie ako tri znaky ako menej významné.

Alternatívnymi technikami na určenie totožnosti slov sú *Soundex*, *Metaphone*, *Levenshtein*. Metóda *Soundex* za totožné považuje aj slová, ktoré podobne znejú. Metóda *Metaphone* používa gramatické pravidlá na konverziu slov. Metóda *Levenshtein* počíta vzdialenosť dvoch reťazcov ako minimálny počet vložení, vymazaní a modifikácií znakov potrebných, aby sa reťazce stali identickými.

Všetky opísané techniky sa môžu aplikovať buď na názov dokumentu alebo na názov a obsah súčasne alebo len na obsah.

4.1.3 Zhodnotenie

MyFreddy je webová pospolitosť zábavy s bohatým obsahom. Dovoľuje jednotlivcom zadávať jej obsah. Od marca 2001 ju navštívilo vyše 30000 používateľov a zozbieralo sa vyše 350000 hodnotení; obsahuje vyše 3000 objektov hodnotenia v 16 kategóriách. Je to vhodný subjekt na testovanie nových myšlienok a algoritmov. Databáza objektov a hodnotení je bohatá, sú to reálne dáta vhodné na výskum v oblasti dolovania dát a kolaboratívnych systémov.

4.2 Predpríprava webových dokumentov na Internete

Základná systémová technika na zníženie oneskorenia prístupu pri výbere v budúcnosti použitých dokumentov sa zakladá na predpovedaní. Pred tým, ako sa k dokumentu pristupuje, sa vo webovom prostredí používa predpríprava.

4.2.1 Predpríprava na rôznych úrovniach

Typická návšteva webovej stránky prechádza cez niekoľko základných fáz: požiadavka DNS, vytvorenie TCP spojenia, HTTP požiadavka a odpoveď. Predpríprava sa môže použiť v predstihu ku každej z týchto fáz.

Predpríprava DNS

Pred vytvorením spojenia na webový server musí klient preložiť časť mena hostiteľa z URL adresy na IP adresu. Webový prehliadač najprv odošle požiadavku DNS na lokálny DNS server. DNS server odpovie priamo zo svojho *cache*, ak pozná požadovanú odpoveď. V prípade, že ju nepozná, nadviaže spojenie s inými DNS servermi na získanie požadovanej IP adresy. DNS server zaznamenáva adresu na TTL (*Time to Live*) sekúnd. Parameter TTL pre príslušného hostiteľa určuje jeho lokálny administrátor. Väčšina súčasných TTL je 24 hodín. Čakanie na odpoveď na požiadavku od DNS servera predstavuje oneskorenie v uspokojení požiadavky používateľa. Na zabránenie tohto oneskorenia môže webový klient iniciovať preklad meno-adresa v predstihu.

Predpríprava TCP spojenia

Pred tým, ako webový klient odošle HTTP požiadavku na server, sa vytvorí TCP spojenie. Vytvorenie TCP spojenia zahŕňa round trip time (RTT) cez sieť, čo je čas predstavujúci interval odoslania malého paketu zo strany klienta na stranu servera a potom späť na stranu klienta zo servera.

Každý smerovač pozdĺž cesty zisťuje IP adresu pre nasledujúci skok a prekladá túto adresu na nízko-úrovňovú MAC adresu. Táto operácia môže vyžadovať komunikáciu s ostatnými smerovačmi. Výsledok operácie môže smerovač predpripraviť na určitú časovú periódu z dôvodu urýchlenia smerovania ďalších IP datagramov na rovnakú cieľovú adresu.

Predpríprava obsahu

Po vytvorení TCP spojenia webový klient pošle požiadavku HTTP na webový server. Oneskorenie počas prijatia HTTP odpovede určuje množstvo faktorov vrátane času potrebného na generovanie odpovede na strane servera, veľkosť odpovede, rýchlosť spojenia medzi serverom a klientom. Klient môže pred používateľom ukryť toto oneskorenie odoslaním HTTP požiadavky v predstihu a predpripraviť si tak odpoveď.

4.2.2 Podmienky predprípravy obsahu

Pri vytvorení výkonnej predprípravy na redukciu oneskorenia treba splniť tri podmienky. Prvou je, že dokument sa môže predpripraviť pred tým, než ho niečo požaduje. Predikcia sa štandardne zakladá na histórii webového prehliadača. Druhou podmienkou je, že predpripravený dokument je v čase požiadavky „čerstvý“. Poslednou podmienkou je, že na predprípravu existuje určitý časový interval pred odoslaním požiadavky.

Informácie z histórie pre predprípravu

Zvyčajne sa ako kritériá pre rozhodovanie, či možno webovú stránku predpripraviť, používajú odvodené z informácie histórie prístupov webového prehliadača. Určenie závislostí webových stránok sa často zakladá na čerstvých prístupových záznamoch; server/klient môže zoskupovať webové stránky so závislosťami navzájom vyššími ako určitý prah pre predprípravu.

Expiračná doba

Keďže platnosť dát môže vypršať pred tým, ako ich niečo požaduje, predpríprava do ďalekej budúcnosti spôsobuje tiež problémy. Niektoré štúdie preukázali, že približne 60% dokumentov má expiračnú dobu rovnú nule. Tieto dokumenty sa preto nedajú použiť na predprípravu, keďže v čase požiadavky prístupu k nim už ich dáta nebudú platné.

Čas pre predprípravu

Na rozhodnutie o predpríprave a predprípravu sa používa čas medzi jednotlivými požiadavkami. Dokument, ktorý je vo fáze predprípravy, sa môže požadovať aj pred časom ukončenia predprípravy. V tomto prípade je výsledok čiastočný úspech. Čas dostupný na predprípravu je čas nečinnosti medzi dvomi súvislými URL požiadavkami toho istého klienta.

4.2.3 Klasifikácia metód predprípravy

Podľa zdroja histórie prístupu použitého na predikciu môžeme rozdeliť metódy predprípravy do štyroch kategórií:

- predpríprava na strane klienta,
- predpríprava na strane proxy servera,
- predpríprava na strane servera,
- kooperatívna predpríprava.

Predpríprava na strane klienta

Najjednoduchší spôsob predprípravy sa zakladá na predikcii založenej na histórii prístupov klienta. Tento spôsob získal množstvo pozornosti z dôvodu jeho potenciálneho výkonového zlepšenia bez nutnosti zmeny webových serverov. Predprípravu na strane klienta môžeme rozdeliť na *pažravú* a *nepažravú* stratégiu. V prípade *pažravej* stratégie môže používateľ špecifikovať len niekoľko základných parametrov, ako množstvo a druh zdrojov na výber predpripravených objektov. Napriek tomu, že tento spôsob predprípravy sa dá implementovať veľmi rýchlo, zvyčajne sa nepoužíva, keďže spôsobuje mimoriadnu réžiu. *Nepažravá* stratégia sa zakladá na histórii prístupov a pokúša sa o predikciu v blízkej budúcnosti pravdepodobne navštívených vnorených odkazov.

Rozsiahle experimenty ukazujú, že pri priemernom zrýchlení prístupu o viac ako 50% sú režijné náklady menej ako 150%.

Predpríprava na strane proxy servera

Inou cestou je vytvárať predikcie nad informáciami zozbieranými na strane proxy servera. V tejto kategórii poznáme dva typy predprípravy: interná predpríprava a externá predpríprava. Pri internej predpríprave sa nepoužíva komunikácia medzi webovým serverom a proxy serverom.

Proxy server neizoluje webové prístupy viacerých používateľov. Práve preto môže často predpovedať, ktorý dokument bude používateľ požadovať. Ak je dokument predpripravený na proxy serveri, proxy server môže zúžitkovať čas nečinnosti na dopravu dokumentu k používateľovi. Keďže v tomto prípade proxy server iniciuje predvýber dokumentu z vlastnej pamäte, nie je nutný žiadny extra internetový prenos.

Štúdie ukazujú, že len zvýšením diskového priestoru na predprípravu na strane klienta môžeme redukovať oneskorenia o 4%. V prípade, že túto techniku kombinujeme s kompresnými technikami, môžeme redukovať oneskorenia o 14,6%. Perfektná a realistická predpríprava kombinovaná z veľkým diskovým priestorom na predprípravu na strane klienta môže redukovať oneskorenia až o 28,6%. Výsledky preukazujú, že predpríprava je efektívna v prípade, že medzi jednotlivými prístupmi toho istého klienta existuje dostatočný časový interval.

Predpríprava na strane servera

V porovnaní s predchádzajúcimi dvomi alternatívami, pri tejto alternatíve dosahujeme vyššiu presnosť, keďže záznamy webového servera obsahujú významne viac informácií v porovnaní s proxy serverom a klientom.

Vo všeobecnosti server predpovedá pravdepodobnosť, že príslušná webová stránka sa neskôr sprístupní a tieto informácie doručuje potom klientovi. Klient po získaní informácie učiní rozhodnutie či stránku predpripraví alebo nie.

Metódy predprípravy na strane servera môžeme približne rozdeliť na dva druhy – PUSH a PULL.

Základnou ideou PUSH metódy je, že servery (a proxy servery) zverejňujú svoje často navštevované stránky. Čím viac stránok sa prehľadáva, tým vyšší je podiel predpripravených stánok, ktoré sa môžu byť rýchlo sprístupniť. Dosahuje sa zlepšenie medzi 3% až 23%.

V prípade metódy PULL poskytuje server pre klientov (alebo proxy servery) rady (informácie o ohodnotení príslušnej stránky) a klient na základe nich môže učiniť rozhodnutie o prípadnej predpríprave stránky. Tieto rady server pripája väčšinou medzi odpovede typu GET. Použitím tejto metódy na zníženie času prístupu približne o 40% spotrebujeme len o 70% viac sieťových prostriedkov.

Kooperatívna predpríprava

Kooperatívna predpríprava predstavuje akúsi kombináciu prechádzajúcich metód. Pri tejto metóde sa využívajú informácie aj zo strany servera, aj zo strany proxy servera. Pri prístupe k požadovanému dokumentu sa postupuje od webového klienta. V prípade, že klient nemá predpripravený požadovaný dokument, požiada oň proxy server. V prípade, ak je pripravený dokument, zobrazí sa používateľov, súčasne sa však pošle informácia o prístupe k dokumentu proxy serveru. Podobne vyzerá situácia medzi proxy serverom a cieľovým serverom.

V prípade použitia tejto metódy ukazujú simulácie, že úspešnosť kooperatívnej predprípravy je o 75% vyššia ako úspešnosť predprípravy len za použitia proxy servera a len o 3% nižšia ako ideálna predpríprava na strane servera.

4.2.4 Predpríprava štruktúry a optimalizácia

Skoro všetky schémy na predprípravu vyžadujú nejaký prediktor na vykonanie niektorých rozhodnutí konštruovaných z prístupových informácií z histórie. Fakty pre túto štruktúru sú takéto:

1. správna predikcia
2. kompaktné dátové štruktúry
3. stabilné vykonávanie

Predikcia na parciálnej zhode

Výsledky výskumu z pospolitosti dátovej kompresie vytvorili niekoľko kontextových modelov na použitie *m* predchádzajúcich symbolov pre predikciu pravdepodobnosti ďalšieho symbolu. Predikcia na parciálnej zhode je jedným z kontextových modelov, ktoré sa aktívne použili vo webovej predikcii. Na základe Markovových predikčných metód prvého rádu pre predikciu súborov sa vďaka výskumom vytvoril graf závislostí, obsahujúci uzly pre všetky kedykoľvek predtým sprístupňované súbory na príslušnom webovom server.

Pravdepodobnosť podmieneného prechodu na stránku X_2 v prípade, že návštevník navštívil stránku X_1 môžeme definovať ako:

$$p(x_2|x_1) = \Pr(X_2 = x_2|X_1 = x_1)$$

Viac všeobecne, použitím Markovovho modelu n-tého rádu môžeme zapísať pravdepodobnosť prechodu na n-tú stránku, ak sme pred ňou navštívili *k=n-1* stránok ako:

$$p(x_n|x_{n-1},...x_{n-k}) = \Pr(X_n = x_n|X_{n-1},...,= X_{n-k})$$

Pre Markovov model 0-tého rádu by sa táto pravdepodobnosť rovnala:

$$p(x_n) = \Pr(X_n)$$

čo je ekvivalentné pravdepodobnosti, že používateľ danú stránku navštívi.

Definujme X_n ako náhodnú premennú, ktorej hodnota predstavuje ktorú stránku navštívi nejaký klient. Pre Markovov model *k*-teho rádu je nestálosť hodnôt X_n a relácia medzi nestálosťou a *k* sa môže určiť analyzovaním entrópie modelov. Entrópia *H(X)* je náhodná premenná definovaná:

$$H(X) = E\left(\log_2 \frac{1}{p(X)}\right) = \sum_{x\in X} p(X)\log_2 \frac{1}{p(X)} = -\sum_{x\in X} p(X)\log_2 p(X)$$

Podmienená entrópia (miera nestálosti X_n, ktorá zostáva aj po tom, čo klient navštívil *k* predchádzajúcich stránok) sa môže definovať takto:

$$H(X_n \mid X_{n-1},..., X_{n-k}) = \sum_{x_n\in X_n} p(x_n)H(X_n \mid X_{n-1} = x_{n-1},..., X_{n-k} = x_{n-k})$$

Z analýzy entrópie sa môže vyvodiť množstvo záverov. Modely nižšieho rádu majú vyššiu pravdepodobnosť nájdenia kandidáta na predprípravu, zatiaľ čo modely vyššieho rádu majú vyššiu presnosť pri ohodnocovaní predpripravených súborov.

Najdlhšie opakujúca sa podpostupnosť

Technika najdlhšie opakujúcej sa podpostupnosti nahliada na problém ako úlohu dolovania dát, kde niektoré cesty sa uvažujú za hluk. Toto reflektuje fakt, že množstvo ciest sa vyskytuje nefrekventovane, často z dôvodu chybnej navigácie.

Navrhli dva čisto spätné hybridné modely na extrakciu najdlhšie opakujúcej sa podsekvencie z prístupovej histórie.

Prvý hybridný model dekomponuje každú postupnosť do série podpostupností dĺžky 2. Napríklad postupnosť ABCD bude mať za výsledok podpostupnosti AB, BC a CD. Tento model sa dá porovnať s Markovovým modelom prvého rádu.

Druhý hybridný model dekomponuje každú postupnosť do množiny podpostupností so všetkými možnými dĺžkami. Napríklad postupnosť ABCD sa rozloží do postupností ABCD, ABC, BCD, AB, BC, CD. Tento model sa dá porovnať s Markovovým modelom *k*-teho rádu.

Predikcia na čiastočnej zhode založená na popularite

Popularita webového URL odkazu sa štandardne charakterizuje počtom prístupov na ňu za časový interval. Táto informácia je veľmi dôležitá v prostredí svetovej pavučiny, keďže 90% požiadaviek prichádza na menej ako 10% dokumentov. Popularita rôznych URL sa môže ohodnotiť raz za čas serverom dynamicky.

Relatívnu popularitu URL môžeme definovať ako:

$$RP = \frac{\text{počet prístupov na URL}}{\text{najvyššia popularita po ceste}}$$

každá URL sa môže potom ohodnotiť 4 stupňami rozloženými na základe $\log_{10}$:

Stupeň 3: RP = (10%, 100%]

Stupeň 2: RP = (1%, 10%]

Stupeň 1: RP = (0,1%, 1%]

Stupeň 0: RP ≤ 0,1%

Pri predikcii na čiastočnej zhode založenej na popularite, Markovov prediktorový strom dynamicky rastie s variabilnou dĺžkou v každom druhu, kde populárne URL adresy môžu byť prvé v množine dlhých vetiev a menej populárne dokumenty môžu byť prvé v množine krátkych vetiev.

4.2.5 Vyhodnotenie výkonnosti predprípravy

Zatiaľ čo predpríprava je efektívna cesta na zmenšenie oneskorenia, výhody predprípravy majú aj svoju cenu. Práve preto treba nájsť určitý kompromis medzi znížením oneskorenia a využitím siete.

Medze redukcie latencie

Predpríprava má značný potenciál na zníženie oneskorenia, ktoré vníma zákazník. Množstvo štúdií skúmalo potenciálne vylepšenie webu, ktoré sa môže dosiahnuť práve predprípravou. Oneskorenia pri prístupe k webovým stránkam môžeme rozdeliť na interné oneskorenia na lokálnych sieťach a externé oneskorenia na globálnych sieťach.

Rôzne štúdie preukázali, že externé oneskorenia tvoria viac ako 88% všetkých oneskorení v rámci siete. Tieto štúdie poukázali aj na niektoré ďalšie fakty. Napríklad pasívna predpríprava s nelimitovaným diskovým priestorom pre predpripravené dáta môže redukovať oneskorenie približne o 26%. Na druhej strane, predpríprava založená na lokálnych informáciách získava 41% pri znižovaní oneskorenia. Pridaním hodnotenia zo strany servera môžeme znížiť oneskorenie až o 57%.

Účinky predprípravy na sieti

Predpríprava vo všeobecnosti zvyšuje prenosovú rýchlosť individuálnych zdrojov, vedie však k zvýšeniu priemernej dĺžky frontov na sieťových prepínačoch. Napriek tomu riadením rýchlosti prenosu môže predpríprava významne zlepšiť sieťový výkon. Riadenie prenosovej rýchlosti sa zakladá na pozorovaní, že počas predprípravy dokumentu nie je nutné využiť plnú rýchlosť dostupnej siete, je lepšie vykonávať predprípravu tak, aby prebiehala čo možno najmenšou rýchlosťou, avšak tak, aby pri požiadavke používateľa bol už príslušný dokument pripravený.

Analýza výhod a nevýhod

Predpríprava sa môže potenciálne využiť v širokom množstve aplikácií. Aj keď, z jej špekulatívnej povahy vyplýva, že môžeme stratiť určitú šírku pásma a kapacitu úložísk, práve tieto straty prinášajú zmenšenie času odozvy. Medzi výhodami a nevýhodami predprípravy treba nájsť čo najvýhodnejší kompromis pre konkrétne využitie.

Predpokladajme, že P je zvýšená početnosť ako daň za predprípravu a D_0 priemerné prístupové oneskorenie bez predprípravy. Za predpokladu, že prístupové oneskorenie je

0 pre predpripravené webovej stránky, a oneskorenie je D_x pre nepredpripravené stránky, priemerná doba oneskorenia prístupu s predprípravou D_n sa môže aproximovať ako:

$$D_n = P.0 + (1-P).D_x = (1-P).D_x$$

Keďže predpríprava je užitočná len v prípade, že $D_n < D_0$, môžeme zapísať

$$D_n / D_0 < 1/(1-P)$$

D_x a D_0 môžeme vypočítať použitím Markovovho modelu prvého rádu takto:

$$D_0 = 1/(1-R_0), D_x = 1/(1-R_x)$$

Definovaním účinnosti predprípravy E ako pomeru zvýšenej početnosti predprípravy a pomeru zvýšenia využitia siete na dosiahnutie danej početnosti z rovnice dostávame:

$$E = \frac{P}{(R_x - R_0)/R_0} > \frac{R_0}{1-R_0}$$

čo implikuje, že E môže byť väčšie ako $R_0 < \{1 - R_0\}$ na zaručenie nižšieho priemerného oneskorenia pri použití predprípravy. Z čoho ďalej môžeme odvodiť:

$$R_0 < E/(1+E)$$

Pre príslušné E takto môžeme určiť maximálne R_0. Predpríprava je užitočná v prípade keď sieťový prenos je malý alebo keď je vysoká účinnosť.

4.2.6 Súvisiace aplikácie

Vyhľadávacie prostriedky

Internetové vyhľadávacie služby sa spoliehajú na vzory správania používateľov pri navštevovaní webových stránok a vďaka nim sa snažia ohodnotiť výsledky vyhľadávania. Príkladom môžu byť prehľadávače ako Google, Yahoo a podobne.

Odporúčacie systémy

Dôležitou súčiastkou pri riešení personalizácie začínajú byť populárne odporúčacie systémy používané napríklad na stránkach `www.amazon.com` a `www.CDNow.com`. Tieto systémy predpovedajú pre klientov položky podľa odporúčaní alebo histórie prístupov ostatných klientov.

4.2.7 Zhodnotenie

Príchod webovej inteligencie odkazuje na systémy a úsilie naučiť sa ťažiť z organizovaných znalostí alebo inteligencie na Internete. Techniky predprípravy z predchádzajúcich častí sú niektoré z cieľov webovej inteligencie. Existuje predpoklad, že techniky predprípravy budú onedlho hrať dôležitú rolu ako časti informačnej a Internetovej technológie.

5 INTELIGENCIA SOCIÁLNYCH SIETÍ

Sociálna sieť je sociálna štruktúra vytvorená z vrcholov, ktoré môžu vo všeobecnosti reprezentovať nejaké indivíduá alebo organizácie. Načrtáva spôsob, akým sú tieto vrcholy prepojené na základe rôznych spoločenských príbuzenstiev od náhodnej známosti po úzky rodinný či pracovný vzťah. Táto časť sa venuje spracúvaniu, manažovaniu a využívaniu inteligencie v sociálnych sieťach, teda narábaniu s informáciami v nich obsiahnutých.

Najskôr sa venujeme manažmentu znalostí v sociálnych sieťach. Po úvode do sociálnych sietí nasleduje klasifikácia webu ako sociálnej siete; ďalej sa venujeme identifikovaniu spoločenstiev na webe, parametrom grafu webu, ak web ponímame ako graf. Potom nasleduje úvaha o tom, či je web fraktálový, teda či časti webu vyzerajú ako miniweby a na záver sa užšie venujeme manažmentu znalostí v sociálnych sieťach.

Druhá časť sa zameriava na algoritmy usporiadania aplikovateľné na sociálne siete. Po úvode do algoritmov usporiadania a do teórie sociálnych sietí nasleduje opis dobre známych algoritmov Pagerank a HITS, po ktorých je uvedený nový algoritmus usporiadania NodeRanking. Tento algoritmus nepotrebuje ku svojej práci celú maticu susednosti grafu – stačí mu len lokálna informácia o grafe, keď si pre každý vrchol pamätáme len jeho susedov. Pokračujeme porovnaním algoritmov usporiadania a experimentami s nimi. Experimenty sa vykonali na vybranej sociálnej sieti a na Klemm-Eguíluzovom modeli webu s 25000 vrcholmi.

5.1 Sociálne (spoločenské) siete – od webu k manažmentu znalostí

Web je veľmi bohatý zdroj informácií. Rôznorodosť autorov, štýlov, motivácií je v kontraste s vyhľadávaním informácií, ktoré je viac riadené a homogénne. Analýza hypertextových odkazov poskytla užitočné nástroje zlepšujúce vyhľadávanie. Výskum štruktúry hypertextových odkazov sa zaoberá teóriou grafov, dolovaním v dátach a manažmentom znalostí.

Stanley Milgram (Wasserman & Faust, 1994) v roku 1967 v USA vo svojej práci postavil základ modernej teórii sociálnych sietí. Jeho experiment bol o doručovaní správy najkratšou cestou medzi dvoma ľuďmi pomocou sprostredkovateľov – priateľov, ktorí sa poznajú. Strednú dĺžku cesty doručenej správy určil na 6 krokov. Teda každá dvojica ľudí v USA je spojená pomocou sociálnej (spoločenskej) siete v 6 krokoch (stupňoch oddelenia). Existuje silná štruktúrna podobnosť medzi webom ako sieťou a sociálnymi sieťami. Autori predpokladajú, že výskum povedie k pokroku v manažmente znalostí.

Použitý zápis: web je súvislý graf, webové stránky sú vrcholmi, hypertextové odkazy sú hranami.

5.1.1 Analýza hypertextových odkazov na webe

Zložitosť webu znemožňuje použiť techniky z manažmentu databáz. Textové vyhľadávacie nástroje fungujú úboho, nezohľadňujú postupy vytvárania obsahu stránok, že stránky neukazujú na ďalšie stránky pomocou hypertextových odkazov bezdôvodne. Prepojenia stránok sú prejavom autorových kontaktov na známych.

HITS algoritmus (Hyper-Link-Induced Topic Search)

Webové stránky sú dvojaké (Kleinberg, 2000):

1. autority (autorské zdroje),
2. rozvetvenia (zoznamy odkazov na zdroje).

Vzťahy medzi stránkami oboch druhov sú vzájomne podporujúce sa.

HITS algoritmus formalizuje intuíciu dobrých vzťahov medzi oboma druhmi stránok do iteratívneho výpočtu:

1. fáza vzorkovania – vytvorí sa koreňový zoznam stránok podľa textového vyhľadávania, ten sa rozšíri do základného zoznamu o stránky dosiahnuteľné priamo cez hypertextové odkazy z koreňového zoznamu stránok,
2. fáza rozposielania ohodnotenia – rozširuje zoznam pomocou hypertextových odkazov, vytvorí sa strom (indukovaný podgraf) zo základného, pre každú vetvu sa počíta ohodnotenie (váha relevancie) pomocou pravidla prepočtu:

$$a_p = \sum_{q:\, q \to p} h_q$$

$$h_p = \sum_{q:\, p \to q} a_q$$

Stránky sú označené ako p, q, váha autority a_p, váha rozvetvenia h_p.

Matematicky sú to lineárne algebrické operácie na incidenčnej matici základného grafu.

Rozšírenia HITS algoritmu

Môže sa stať, že HITS algoritmus sledovaním hypertextových odkazov prejde do príbuznej alebo všeobecnejšej témy vo výsledku vyhľadávania. Preto sa začali tvoriť rôzne heuristiky:

1. analýza textu opisujúceho odkaz, rozdelenie veľkého zoznamu odkazov na menšie relevantné časti, priemerovanie príspevku viacnásobných odkazov,
2. *Pagerank* (Google) – ohodnocovanie nezávislými dopytmi na najviac navštevované stránky,
3. *Salsa* – obdoba HITS algoritmu s dvoma maticami.

5.1.2 Komunity (spoločenstvá) na webe

Komunita na webe je skupina stránok so spoločným obsahom. Niektoré komunity sú dostupné ako špeciálne služby: *newsgroups*, *email groups*, *mailinglists*, osobné webové

stránky, portály a iné. Zaujíma nás automatické rozpoznanie komunít. Úspešnosť HITS algoritmu záleží na tom, či komunita obsahuje hustú sieť hypertextových odkazov na autorské zdroje. Teda ide o rozpoznanie hustých bipartitných grafov, kde podgrafy sú kompletnými bipartitnými grafmi. Technika na identifikovanie komunít sa volá *trawling* (chytanie rýb do siete). Pre veľkú sieť je táto technika veľmi časovo náročná.

V inej práci sa komunity definujú ako zoskupenie stránok, ktoré majú viac odkazov na členov komunity ako odkazov mimo komunity.

5.1.3 Prepojenie a veľkosť siete

Je web dobre prepojený alebo sa dá rozdeliť na malé časti? Je možné dostať sa na akúkoľvek stránku z akejkoľvek stránky len pomocou hypertextových odkazov? V práci (Albert et al., 1999) sa spomína web ako digraf s priemerom 19. Niektoré práce demonštrovali zákon exponentu pre stupne vrcholov. Broder (Broder et al., 2000) študoval časť webu okolo stránky Altavista.

Analýza prepojenia webu má tieto výsledky

Web sa dá rozdeliť na 4 približne rovnako veľké časti:

- silne prepojená časť, centrum webu, kde sú všetky veľké a známe stránky, vzájomne sa na seba odkazujúce,
- slabo prepojená časť jednosmerne do silne prepojenej časti (nové stránky),
- slabo prepojená časť jednosmerne zo silne prepojenej časti (firemné stránky),
- ostatné neprepojené stránky (extranet, neplatné odkazy).

Teda internet nie je celý dobre (silno) prepojený, ani sa nedá celý rozdeliť na malé časti.

Priemer (šírka) webu

Striktne teoreticky je nekonečno, lebo existujú stránky, ktoré nie sú dosiahnuteľné cez hypertextové odkazy. Pre dosiahnuteľnú časť webu je zistená priemerná hodnota 16. Ak neberieme do úvahy smer odkazov, potom je to 7, čo je sieť malého rozsahu. Taká sieť malého rozsahu sa podobá sociálnym sieťam.

5.1.4 Fraktálová prirodzenosť webu

Je web fraktálový? Teda vyzerajú časti webu ako miniweby? Skúmala sa štruktúra webu cez teóriu grafov. Podgrafom bola prisúdená veľkosť 10% celého grafu. Časti webu pritom majú obsahovať aspoň 10 000 stránok.

Výsledky potvrdzujú hypotézu. Podľa experimentálnych zistení aj skupina stránok s podobným obsahom má vlastnosti celého webu. Potom takéto miniweby sú prepojené navigačnou chrbticou (vyhľadávačmi, katalógmi). Fraktálová samo-podobnosť je aj v sociálnych sieťach.

5.1.5 Sociálne siete pre manažment znalostí

Zovšeobecnenie analýzy hypertextových odkazov na webe:

- zmysel odkazu (organizácie, online nákupy, záujmy)
- cez analýzu správania môže byť manažment znalostí úspešnejší ako vyhľadávanie (odporúčania, experti, komunity, skupiny záujmov)

Spomenuté algoritmy (HITS, komunity) predpokladajú vzťah medzi entitami jedného typu. Otázka do budúcnosti je v tom, ako kombinovať viaceré typy entít.

Firemné aplikácie manažmentu znalostí vyžadujú, aby systém kombinoval implicitné odkazy (dvaja používatelia pozerajú ten istý dokument) a explicitné odkazy (dvaja používatelia v tej istej pracovnej skupine pravdepodobne sa budú zaujímať o tú istú informáciu). Otázka do budúcnosti je v tom, ako skombinovať tieto dva prístupy.

Ďalší smer výskumu je informačná bezpečnosť. Informačná bezpečnosť je širší pojem ako dátová bezpečnosť. Ako sa mení správanie používateľa, keď poskytuje informácie o sebe do systému? Vyhľadávače sa dajú vylepšiť tak, že si modelujú používateľa, jeho záujmy, požiadavky na vyhľadávanie. Ako to môže prospieť ostatným používateľom?

Informačná bezpečnosť vo firemných systémoch zahŕňa aj zoznam prístupových práv používateľov alebo skupín používateľov k dokumentom.

Firemný manažment znalostí

Táto aplikácia pomáha zhodnotiť informácie od jednotlivcov alebo skupín jednotlivcov a sprístupňuje tieto znalosti iným jednotlivcom alebo skupinám jednotlivcov. Teda zaoberáme sa sieťami, ktoré sú vo vnútri alebo okolo firiem. Skúmaná sieť zahŕňa rôzne typy entít – webové stránky, elektronickú poštu, dokumenty, organizácie, komunity, expertné domény, ľudí.

Vynárajú sa otázky:

- Kto sú experti v mojej organizácii v nejakej oblasti? Skúmame sociálne siete ľudí a elektronickú komunikáciu medzi nimi.
- Existujú v organizácii časti, ktoré duplikujú činnosť? Skúmame siete dokumentov, procesy vytvárania dokumentov.
- Existujú iní v mojej organizácii, ktorí už robili podobné činnosti, aké robím ja? Skúmajú sa individuálne profily, nie sieťový model.
- Ktoré oddelenie mojej organizácie (ktorí ľudia) by ma mali najviac zaujímať? Skúma sa neformálna komunikácia v sociálnej sieti.

Existujú základné procedurálne otázky, na ktoré nemáme uspokojivé odpovede:

- Ako efektívne organizácia spracúva a používa znalosti a aká kvantitatívna mierka to zachytí?
- Ako sa môže merať dopad vybraného prístupu k manažmentu znalostí?
- Akú mierku dať dopadu poznania tradičnej merateľnej výkonnosti?

5.1.6 Zhodnotenie

Web je veľká sociálna sieť. Skúmanie jej štruktúry vyústilo do vytvorenia efektívnych algoritmov pre vyhľadávanie a dolovanie informácií. Web má súčasne blokovú a súčasne fraktálovú štruktúru. Nakoniec sme spomenuli niekoľko závažných otázok z oblasti firemného manažmentu znalostí.

5.2 Algoritmus usporiadania na určenie miery dobrej povesti alebo relevancie založený na topológii grafu

V oblasti informačných technológií sú posledné roky poznačené obrovským nárastom celosvetovej pavučiny. Množstvo dostupných informácií je rozsiahle, no nepoužiteľné bez predchádzajúceho predspracovania. Usporiadanie webových stránok podľa dôležitosti sa zdá byť jedným z možných riešení: umožňuje to veľké kvantum informácie nejako zoradiť.

Prvé pokusy o výpočet usporiadania sa zameriavali na textový obsah webových stránok. Tento spôsob analýzy sa ukázal byť nedostatočný na získavanie usporiadania a ďalším krokom bolo použitie odkazov medzi jednotlivými webovými stránkami. Tento prístup sa ukázal byť vhodnejší: odkazy sú dobrým zdrojom informácií na vygenerovanie usporiadania. Je to posun od využívania informácie obsiahnutej vo webovej stránke (teda obsahu vrcholu, ak si web predstavíme ako graf) ku využitiu informácie o štruktúre odkazov medzi stránkami (teda štruktúre grafu).[7]

Je známych niekoľko algoritmov, ktoré využívajú ponímanie webu ako grafu. Autori sa zaoberajú metódou, ktorá každému vrcholu priraďuje nejaké skóre. Vychádzajú z princípu, že vrchol *i* v danom grafe je dôležitejší, t.j. má vyššie skóre a lepšiu pozíciu v usporiadaní, ako vrchol *j*, ak je spojený s vrcholmi, ktoré majú skóre vyššie než vrcholy spojené s *j*. Inými slovami, skóre, rank, dôležitosť, kvalita alebo autorita vrcholu *i* vychádza z počtu vrcholov, ktoré majú odkaz na vrchol *i* a z kvality týchto vrcholov. Táto metóda je teda založená na myšlienke, že miera, do akej sa možno spoľahnúť na kvalitu vrcholu, sa prenáša odkazmi medzi vrcholmi. Na tejto myšlienke sú založené aj známe algoritmy usporiadania HITS a Pagerank.

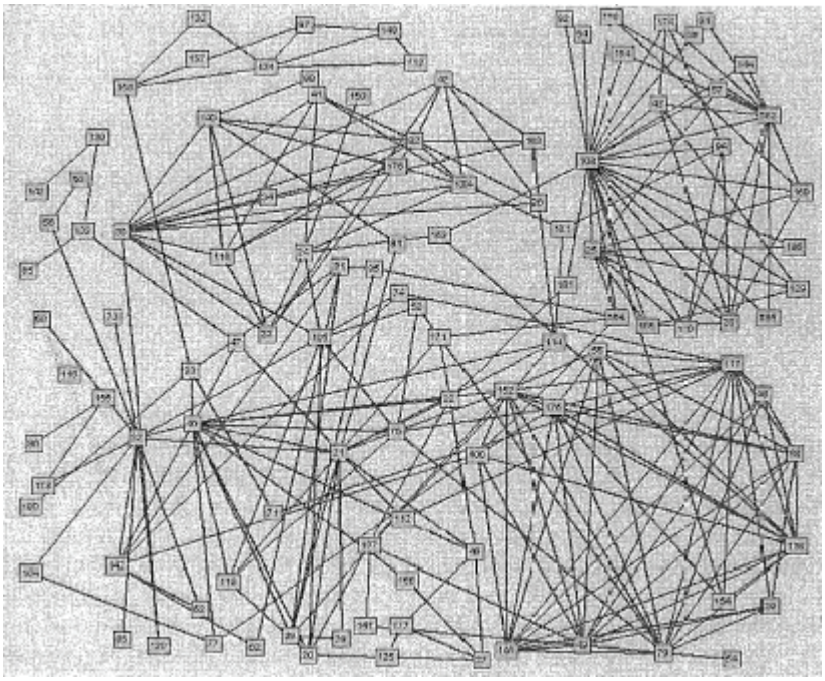
Predmetom práce je nový algoritmus na určovanie usporiadania NodeRanking, ktorý by mal na rozdiel od algoritmov HITS a Pagerank pracovať v grafe s využitím len lokálnej informácie, t.j. pri tomto algoritme nie je nutná znalosť celého grafu. Tento nový prístup autori otestovali na dvoch druhoch sietí: na sociálnej sieti reálnej komunity ľudí a na sieťach, ktoré majú niektoré topologické vlastnosti webu.

5.2.1 Sociálne siete

Sociálna sieť je reprezentáciou vzťahov existujúcich vo vnútri komunity ľudí. Sociálne siete sú reprezentované grafmi. Z umiestnenia nejakého daného člena komunity vo vnútri sociálnej siete môže byť odvodený stupeň dobrej povesti člena v komunite. Experti dobre známi a vysoko oceňovaní väčšinou členov komunity sú ľahko identifikovateľní ako vrcholy s vysokou súvislosťou - prepojenosťou v grafe.

Prípadová štúdia autorov je založená na sociálnej sieti UPC Softvérového oddelenia (pozri obrázok 5-1).

[7] Navyše, ak si predstavíme odkaz medzi dvomi webovými stránkami ako nejaký vzťah medzi dvomi indivíduami, tak do popredia vystúpi akési sociálne povedomie. V tomto zmysle hocijaká sociálna sieť, nielen web, poskytuje informáciu o usporiadaní indivíduí obsiahnutých v jej štruktúre.



Obrázok 5-1. Časť neorientovanej sociálnej siete UPC Softvérového oddelenia (zdroj: Zhong et al., 2003).

Túto sociálnu sieť vygenerovali automaticky softvérom, ktorým je multiagentový systém *NetExpert*. Tento systém podporuje zdieľanie znalostí vo vedeckej komunite. Sociálnu sieť vytvorili na základe analýzy domovských stránok členov komunity, autorstva dokumentov, podieľania sa na projektoch a pod.

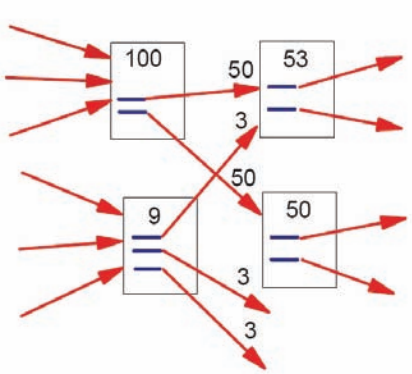
5.2.2 Algoritmy usporiadania

Najprv uvedieme stručný prehľad známych algoritmov Pagerank a HITS; potom nový algoritmus NodeRanking.

Pagerank

Zakladá sa na myšlienke, že poradie vrcholu v ohodnotení je vysoké, ak naň odkazujú vrcholy s vysokým ohodnotením. Postup určovania ohodnotenia webových stránok na základe odkazov medzi nimi znázorňujú nasledujúce dva obrázky (pozri obrázok 5-2 a 5-3). Pri takomto iteratívnom prístupe k určovaniu ohodnotenia môže dôjsť ku tzv. z*ahlteniu ohodnotenia*, keď ohodnotenie stránok rastie do nekonečna. Táto situácia môže nastať v prípade cyklických odkazov medzi stránkami (pozri obrázok 5-4).

Na vyhnutie sa takýmto situáciám sa prijala stratégia tzv. náhodného chodca (*random walker*), ktorý sa náhodne „prechádza“ po grafe a vždy po prejdení hrany (odkazu) upraví ohodnotenie vrcholu, do ktorého vstúpi, tak, ako to bolo znázornené na predchádzajúcich troch obrázkoch.



Obrázok 5-2. Určovanie ohodnotenia webových stránok na základe odkazov medzi nimi (zdroj: Page et al., 1998).

Nové v tejto stratégii je, že chodec môže kedykoľvek s istou pravdepodobnosťou skočiť na hociktorý iný vrchol, nielen sa presúvať po hranách. Práve týmto sa dá predísť zahlteniu ohodnotenia (nekonečné slučky možno opustiť). Keby sa to malo napísať presne matematicky v reči matíc, tak by to vyzeralo takto:

L je matica susednosti grafu, *P* je stochastická matica odvodená od *L*

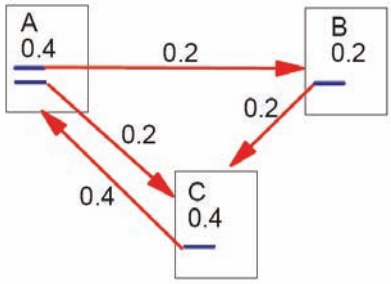
$$p_{ij} = \frac{l_{ij}}{\sum_k l_{ik}}$$

e je pravdepodobnosť skoku 0<=*e*<=1. Konvergencia algoritmu *Pagerank* závisí od parametra *e*. Mal by byť z intervalu <0,1; 0,2>. Odporúčaná hodnota je 0,15. Prechodovú maticu, na základe ktorej *Pagerank* pracuje, možno potom vyjadriť v tvare

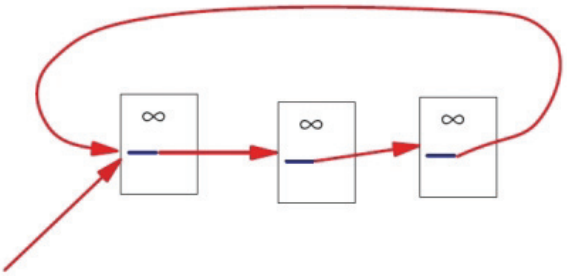
$$M = e(\frac{1}{N} JJ^T) + (1-e)P$$

kde *N* je počet vrcholov v sieti a $J = (11...1)^T$ je jednotkový vektor.

Výsledné ohodnotenie vrcholov potom dostaneme ako hlavný vlastný vektor (vlastný vektor s najväčším vlastným číslom) matice M^T transponovanej ku prechodovej matici *M* zadefinovanej vyššie.



Obrázok 5-3. Určovanie ohodnotenia webových stránok pri cyklickej štruktúre odkazov medzi nimi (zdroj: Page et al., 1998).



Obrázok 5-4. Zahltenie ohodnotenia. Ohodnotenie stránok nebude konvergovať, pretože bude rásť do nekonečna (zdroj: Page et al., 1998).

HITS

V tomto algoritme každý vrchol *i* má priradené skóre *rozvetvenia* y_i a skóre *autority* x_i. Kľúčovou myšlienkou je, že na vrchol s vysokým skóre *autority* odkazuje veľa vrcholov s vysokým skóre *rozvetvenia* a naopak, vrchol s vysokým skóre *rozvetvenia* odkazuje na veľa vrcholov s vysokým skóre *autority*. Inak povedané, ak má vrchol vysoké skóre *rozvetvenia*, vypovedá to o kvalite vrcholov, na ktoré odkazuje a ak má vysoké skóre *autority*, hovorí to o kvalite jeho samotného.

Takže možno zadefinovať dve operácie: *I* (aktualizuje skóre *autority*) a *O* (aktualizuje skóre *rozvetvenia*)

$$I : x_j \leftarrow \sum_{k:(k,j)\in E} y_k \qquad O : y_j \leftarrow \sum_{k:(j,k)\in E} x_k$$

Ak to zapíšeme pomocou matice susednosti *L*, dostaneme

$$x \leftarrow L^T y \qquad y \leftarrow Lx$$

Po vzájomnom dosadení dostaneme iteratívne predpisy

$$x^{(t+1)} = L^T L x^{(t)} \qquad y^{(t+1)} = L L^T y^{(t)}$$

pre nejaké dané $x^{(0)}$ a $y^{(0)}$. Hľadaným skóre *autority* bude teda potom hlavný vlastný vektor matice $L^T L$.

Algoritmus NodeRanking

Je to nový algoritmus na určovanie rankingu vrcholov v grafe. Opäť každý vrchol v grafe má priradený stupeň autority. Na začiatku práce algoritmu majú všetky vrcholy rovnakú autoritu. Po zbehnutí algoritmu výsledná autorita vrcholu môže byť použitá na odvodenie kvality, dobrej povesti, reputácie vrcholu v grafe. Autorita vrcholu *i* je počítaná ako funkcia autorít vrcholov odkazujúcich na vrchol *i*.

Podobne ako algoritmy HITS a Pagerank aj tento algoritmus pracuje na orientovanom grafe. Na rozdiel od nich ale dokáže spracovať aj ohodnotené grafy, čo znamená, že každej hrane je priradená kladná váha.

V orientovanom grafe budeme hrany nazývať vychádzajúcimi z vrcholu a vstupujúcimi do vrcholu. Nasledovníkom vrcholu budeme nazývať ten vrchol, do ktorého sa môžeme dostať vychádzajúcou hranou. Predchodcom vrcholu *i* nazývame ten vrchol, z ktorého sa môžeme dostať do vrcholu *i* vstupujúcou hranou.

Hlavnou myšlienkou algoritmu je, že autorita vrcholu je posúvaná do nasledovníka cez vychádzajúcu hranu. Algoritmus je:

```
do
     n = getNode(g)
     do
            passAuthority(n)
            nnew = getNextNode (n, g)
            n = nnew
     while (nnew != null)
while (!converge())
```

Jednotlivé funkcie si vyžadujú ďalšie vysvetlenie:

getNode(Graph g) vráti náhodne vybratý vrchol

getNextNode(Node n, Graph g) vráti nasledovníka *m* vrcholu *n*. Tento vrchol sa vyberie s pravdepodobnosťou vypočítanou na základe váhy hrany z *n* do *m*.

$$\Pr_{vyber}(n \to m) = \frac{vaha_{n \to m}}{\sum_{\forall l:(n,l) \in E} vaha_{n \to l}}$$

môžu nastať dva prípady, keď sa cesta preruší a *getNextNode* vracia prázdny vrchol:

- prvý prípad, keď algoritmus dorazí k vrcholu, ktorý mal návštevu v predchádzajúcich *k* krokoch (*k* je parameter algoritmu)
- druhý prípad, keď istá hodnota pravdepodobnosti skoku sa dosiahne – keď náhodný chodec s pravdepodobnosťou $\Pr_{skok}$ preruší cestu (pri vrcholoch s menším počtom vychádzajúcich hrán je pravdepodobnosť prerušenia cesty vyššia)

$$\Pr_{skok}(n) = \frac{1}{\#vychadzajucich\_hran(n) + 1}$$

passAuthority(Node x) priraďuje časť autority vrcholu *x* všetkým jeho nasledovníkom

$$\Delta autorita(y) = \frac{\Pr_{vyber}(x \to y) \times autorita(y)}{F_{nc}^{y}}$$

kde *autorita(y)* je autorita vrcholu *y*, F_{nc}^{y} je normalizačný faktor. Bez neho by hodnoty autority rástli do nekonečna. Faktor je pridružený ku čiastkovej ceste náhodného chodca. Faktor častejšie navštevovaných vrcholov rastie rýchlejšie tak, aby sme mohli zaručiť konvergenciu autority ku konečnej hodnote.

converge() je funkcia na testovanie konvergencie algoritmu na všetkých vrcholoch grafu. $\Delta autorita(n)$ sa blíži ku nule, kvôli použitiu faktora F_{nc}^{n}. Keď pre nejaký vrchol sa splní podmienka $\Delta autorita(n) < \varepsilon$ (ε je nastaviteľný parameter algoritmu), tak vrchol sa označí za stacionárny. Keď všetky vrcholy sú stacionárne, algoritmus končí.

Porovnanie algoritmov

HITS a Pagerank počítajú ohodnotenie na základe matice susednosti grafu. To znamená, že matica susednosti musí byť známa pred spustením algoritmu. Naviac, pri každom aktualizovaní ohodnotenia je nutný synchronizačný krok: stav všetkých vrcholov musí byť známy na to, aby sme mohli aktualizovať stav jediného z nich.

To neplatí v prípade algoritmu NodeRanking, keď sa využíva len lokálna informácia o grafe, nie celý graf. Pre každý vrchol si stačí pamätať len vrcholy s ním susedné.

V prípade algoritmu NodeRanking je pravdepodobnosť skoku odlišná pre rôzne vrcholy – závisí od počtu vychádzajúcich hrán. Pri algoritme *Pagerank* táto pravdepodobnosť je stanovená na 0,15 pre celý graf.

5.2.3 Experimenty s usporiadaním

Určenie miery dobrej povesti v sociálnych sieťach

Algoritmy *Pagerank*, *HITS* a *NodeRanking* boli aplikované na sociálnej sieti vytvorenej z 34 vrcholov náhodne vybraných zo siete UPC Softvérového oddelenia. Usporiadanie určené algoritmom *NodeRanking* je označený $Rank_{NodeRanking}$. *Pagerank* a *HITS* algoritmy pracujú na neohodnotených grafoch, no sociálna sieť UPC je ohodnotená, preto sme uvažovali modifikácie týchto algoritmov, ktoré pracujú na grafoch ohodnotených.

Tabuľka 5-1. Korelačné koeficienty medzi referenčnými usporiadaniami $Rank_{cite}$ a $Rank_{cite\text{-}self}$ a usporiadaniami získanými jednotlivými algoritmami.

Korelačný koeficient	$Rank_{cite}$	$Rank_{cite\text{-}self}$
$Rank_{cite}$	1,0	0,983
$Rank_{cite\text{-}self}$	0,983	1,0
$Rank_{NodeRanking}$	0,687, $s=8{,}6.10^{-4}$	0,621, s=0,011
$Rank_{PageRank(w)}$	0,535	0,486
$Rank_{PageRank}$	0,521	0,495
$Rank_{HITSAuth(w)}$	0,412	0,383
$Rank_{HITSAuth}$	0,342	0,323

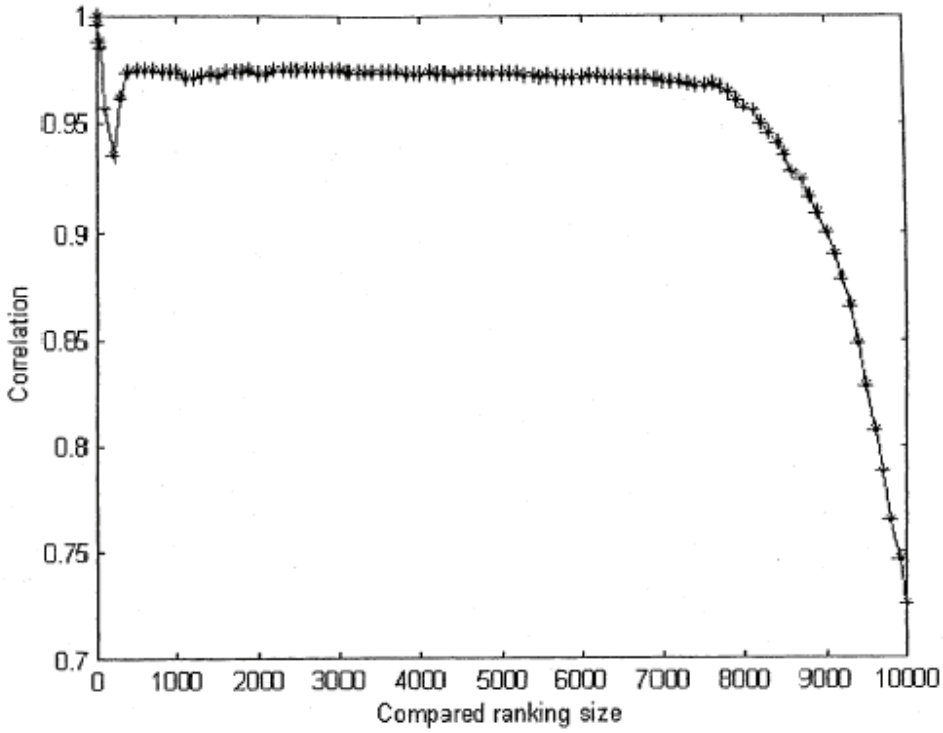
$Rank_{PageRank}$, $Rank_{HITSAuth}$, $Rank_{PageRank(w)}$, $Rank_{HITSAuth(w)}$ sú postupne usporiadania určené algoritmami *Pagerank*, *HITS*, *Pagerank* na ohodnotenom grafe, *HITS* na ohodnotenom grafe. Komunita ľudí vyjadrená sociálnou sieťou UPC je skupina vedcov a tak citačné indexy získané zo CiteSeeru boli referenčnými hodnotami. Vypočítali sa 2 usporiadania: $Rank_{cite}$, v ktorom boli výskumníci usporiadaní podľa počtu citácií a $Rank_{cite\text{-}self}$, kde boli usporiadaní podľa počtu citácií bez citácií samého seba. Tabuľka 5-1 ukazuje korelácie medzi referenčnými usporiadaniami $Rank_{cite}$ a $Rank_{cite\text{-}self}$ a ostatnými. Je vidieť, že nezávisle od referenčného usporiadania, *NodeRanking* dáva lepšie výsledky ako *HITS* a *Pagerank*. Je tiež zaujímavé, že *HITS* dáva lepšie výsledky na ohodnotených grafoch než na neohodnotených.

Stredná pravdepodobnosť skoku pri algoritme *NodeRanking* je 0,5314. Je to tým, že v tejto sociálnej sieti existuje veľa vrcholov so žiadnymi alebo s málo vychádzajúcimi hranami. A tak pevná pravdepodobnosť skoku 0,15 pri algoritme *Pagerank* nie je najvhodnejšia. Keď bola táto pevná hodnota zvýšená na 0,5314, tak *Pagerank* dával podobné výsledky ako *NodeRanking*

Určenie relevancie webových stránok

V tomto experimente sa použil Klemm-Eguíluzov model webu s 25000 vrcholmi. Skúmala sa korelácia medzi usporiadaniami získanými algoritmami *Pagerank* a *NodeRanking*. Ako ukazuje obrázok 5-5 táto korelácia je vysoká.

Na *x*-ovej osi je zobrazený rozsah počtu vrcholov, na ktorých sa sledovala korelácia. Korelácia poradia prvých 10 vrcholov je 0,9964; pri prvých 100 vrcholoch je to hodnota 0,9564; pri prvých 9000 vrcholoch bola odsledovaná hodnota 0,8892.



Obrázok 5-5. Ukazuje koreláciu medzi usporiadaniami získanými algoritmami Pagerank a NodeRanking (zdroj: Zhong et al., 2003).

5.2.4 Zhodnotenie

V práci je uvedený *NodeRanking* – nový algoritmus na určenie usporiadania vrcholov v grafe. Výsledky algoritmu sa porovnávali s usporiadaniami získanými algoritmami *Pagerank* a *HITS* na dvoch grafoch s rôznymi topologickými vlastnosťami: na sociálnej sieti a na modeli webu.

Porovnaním algoritmov na sociálnej sieti sa zistilo, že *NodeRanking* dáva lepšie výsledky ako *Pagerank* a *HITS.* Výsledky získané v experimente na modeli webu zase ukázali, že medzi algoritmami *NodeRanking* a *Pagerank* je vysoká korelácia.

POUŽITÁ LITERATÚRA

AGRAWAL, R. – IMIELINSKI, T. – SWAMI, A. (1993). Database mining: a performance perspective. IEEE *Trans. Know. Data Eng.,* Vol. 5, pp. 914-925.

AGRAWAL, R. – SRIKANT, R. (1994). Fast algorithms for mining associations rules. In: J.B. Bocca, M.Jarke. C.Zaniolo (eds.), *Proc. of 20th VLDB Conference,* Santiago de Chile, 12-15 September 1994, Morgan Kaufmann, pp. 487-499.

AGRAWAL, R. – SRIKANT, R. (1995). Mining sequential patterns. In: P.S.Yu, A.L.P. Chen (eds.), *Proc. 11th Int. Conf. On Data Engineering,* Taipei, March 6-10 1995, IEEE Computer Society, pp. 3-14.

ALBERT, R. – JEONG, S. H. – BARABASI, A. L. (1999). Diameter of the World-Wide Web, In: *Nature*, Vol. 401, pp. 130-131.

ANKOLEKAR, A. – BURSTEIN, M. – HOBBS, J. R. ET AL. (2002). DAML-S: Web Service Description for the Semantic Web. In: *The First International Semantic Web Conference (ISW*C).

ANTONIOU, G. – VAN HARMELEN, F. (2004) *A Semantic Web Primer (Cooperative Information Systems)*. The MIT Press.

ASSADI, H. (1999). Construction of a regional ontology from text and use its use within a documentary system. In: N. Guarino (ed.) *Formal Ontology in Information Systems, Proc.FOIS – 98,* Trento, Italy, pp.236-249.

BERNERS-LEE, T. – HENDLER, J. – LASSILA, O. (2002). The Semantic Web, Scientific American, 05.2005, In: D. Fensel, J. Hendler, H. Lieberman, W. Wahlster (eds.) *Spinning the Semantic Web*, MIT Press, Boston.

BRIJS, T. – SWINNEN, G. – VANHOOF, K.ET AL. (1999). Using association rules for product assortment decisions: a case study. In: *Proc. 5th Int. Conf .on Knowledge Discovery and Data Mining,* San Diego, 15-18 August 1999, ACM Press, pp. 254-260.

BRIN, S. – MOTWANI, R. - SILVERSTEIN, C. (1997a). Beyond market baskets: generalizing association rules to correlations. In: J.Peckham (ed.) *Proc.ACM SIGMOD Int. Conf. on Management Data,* Tucson, 13-15 May 1997, ACM Press, pp. 265-276.

BRIN, S. – MOTWANI, R – ULLMAN, J. ET AL. (1997b). Dynamic itemset counting and implication rules for market baset data. In: J.Peckham (ed.) *Proc. ACM SIGMOD Int. Conf. on Management Data,* Tucson, 13-15 May 1997, ACM Press, pp. 255-264.

BRIN, S. – PAGE, L. (1998). The anatomy of a large-scale hypertextual web search engine. In: *WWW98*, Brisbone, Australia, pp. 107-117.

BRODER, A. – KUMAR, R. – MAGHOUL, F. ET AL. (2000). Graph structure in the Web, In: *Proc. 9th World-Wide Web Conf./Comp. Networks*, Vol. 33, No. 1-6, pp. 309-320.

BRYSON, J. J. – MARTIN D. – MCILRAITH, S. I. ET AL. (2003). Agent-Based Composite Services in DAML-S: The Behavior-Oriented Design of an intelligent semantic web. In: N. Zhong, J. Liu, Y, Yao (eds.) *Web Intelligence*, Springer , pp. 37–58.

BRYSON, J. J. – STEIN, L. A. (2001). Modularity and design in reactive intelligence. In: *Proceedings of the 17th International Joint Conference on Artificial Intelligence*, Morgan Kaufmann, pp. 1115-1120.

CHAU, M. – CHEN, H. (2003). Personalized and Focused Web Spiders. In: N. Zhong, Y. Yao, J. Liu (eds) *Web Intelligence*, part 10, Springer, pages 197-217, ISBN 3540443843.

CHEN, H. (1995). Machine learning for information retrieval: Neural networks. In: *Journal of the American Society for Information Science*, Vol. 46, No. 3, pp. 194-216.

CHEN, H. – CHAU, M. – ZENG, D. (2002). CI Spider: A Tool for Competitive Intelligence on the Web. In: *Decision Support Systems*, Vol. 34, No. 1, pp. 1-17.

CHEN, H. – CHUNG, Y. – RAMSEY, M. ET AL. (1998). A Smart Itsy Bitsy Spider for the Web. In: *Journal of the American Society of Information Science*, Vol. 49, No. 7, pp. 604-618.

CHEN, H. – FAN, H. – CHAU, M. ET AL. (2001). MetaSpider: Meta-searching and Categorization on the Web. In: *Journal of the American Society for Information Science and Technology*, Vol. 52, No. 13, pp. 1134-1147.

DE BRA, P. – POST, R. D. (1994). Information retrieval in the World-Wide Web: Making client-based searching feasible. In: *Computer Networks and ISDN Systems*, Vol. 27, No. 2, pp. 183-192.

FAURE, D. – NEDELLEC, C. (1998). A corpus – based conceptual clustering method for verb frames and ontology acquisition. In: *LREC Workshop on Adapting Lexical and Corpus Resources to Sublanguages and Applications,* Granada, Spain.

FENSEL, D. (2001). Ontologies: Silver bullet for knowledge management and electronic commerce, Springer, Berlin.

FENSEL, D. – HORROCKS, I. – VAN HARMELEN, F. ET AL. (2000). OIL in a nutshell, In: R. Dieng et al. (eds.) Knowledge Acquisition, Modeling, and Management, *Proc. the European Knowledge Acquisition Conference (EKAW-2000)* LNAI, Springer, p. 1937.

FENSEL, D. – HORROCKS, I. – VAN HARMELEN, F. ET AL. (2001). OIL: Ontology infrastructure to enable the Semantic Web, In: *IEEE Intelligent System*, Vol. 16, No. 2.

FIKES, R. E. – HART, P. E. – NILSSON, N. J. (1993). Learning and executing generalized robot plans. In *Readings in knowledge acquisition and learning: automating the construction and improvement of expert systems*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, pp. 485–503.

HAHN, U. – SCHNATTINGER (1998). Ontology engineering via text understanding. In: *IFIP´98 – Proc. The 15th World Computer Congress,* Vienna and Budapest.

HASTINGS, P. M. (1994). *Automatic Acquisition of Word Meaning from Context.* PhD. Thesis, University of Michigan.

HEARST, M. A. (1992). Automatic acquisition of hyponyms from large text corpora. In: *Proc. The 14th International Conference on Computational Linguistics.* Nantes, France.

HEARST, M. A. – SCHÜTZE, H. (1993). Customozing a lexicon to better suit a computational task. In: *Proc. of the ACL-SIGLEX Workshop on Acquisition of Lexical Knowledge from Text,* Columbus, OH, USA.

HOOBS, J. (1993). The generic information extraction system. In: *Proceedings of the fifth Message Understanding Conference (MUC-5),* Morgan Kaufmann.

KAUFMAN, L. – ROUSSEEUW, P. J. (1990). *Finding Groups in Data: An Introduction to Cluster Analysis,* John Wiley.

KLEINBERG, J. (1997). *Authoritative sources in a hyperlinked environment*. Technical Report RJ 10076, IBM.

KLEINBERG, J. M. (2000). *Authoritative sources in a hyperlinked environment.* ACM, Vol. 46, No. 5, pp. 604-632.

LAIRD, J. E. – ROSENBLOOM, P. (1996) The evolution of the Soar cognitive architecture. In: D. M. Steier and T. M. Mitchell (eds) *Mind Matters: A Tribute to Allen Newell*, Lawrence Erlbaum Associates, Inc., Mahwah, New Jersey, pp. 1-50.

MAANNING, C. D. – SCHUETZE, H. (1999). *Foundations of Statistical Natural Language Processing*, MIT Press, Cambridge, MA.

MAEDCHE, A. – STAAB, S. (2000). Discovering conceptual relations from text. In: *ECAI – 2000 – Proc.the 13th European Conference on Artificial Intelligence,* IOS Press, Amsterdam.

MAEDCHE, A. – STAAB, S. (2001). Ontology learning for the semantic Web. In: *IEEE intelligent Systems*, Vol. 16, No. 2.

MAES, P. (1990). A bottom-up mechanism for behavior selection in an artificial creature. In: *Proceedings of the first international conference on simulation of adaptive behavior on From animals to animats*, Cambridge, MA, USA, MIT Press, pp. 238-246.

MCDERMOTT, D. ET AL. (1998). PDDL – the planning domain definition language.

MILLER, R. – BHARAT, K. (1998). SPHINX: A Framework for Creating Personal, Site-SpecificWeb Crawlers. In *Proceedings of the Seventh International WWW Conference*, pp. 161-172.

MORIN, E. (1999). Automatic acquisition of semantic relations between terms from technical corpora. In: *Proc.the Fifth International Congress on Terminology and Knowledge Engineering (TKE´99).*

NILSSON, N. J. (1994) Teleo-Reactive Programs for Agent Control. In: *Journal of Artificial Intelligence Research*, Vol. 1, pp.139-15.

PAGE L., ET AL. (1998) The Pagerank Citation Ranking: Bringing Order to the Web. Technical Report, Stanford University.

PEREIRA, F. – TISHBY, N. – LEE, L. (1993). Distributional Clustering of English Words. In: *Proc. The ACL – 9*, pp. 183-199.

RESNIK, P. S. (1993). *Selection and Information: A Class – based Approach to Lexical Relationship,* PhD thesis, University of Pennsylvania.

STAAB, S. – SCHNURR, H. – P. – STUDER, R. ET AL. (2001). Knowledge processes and ontologies. In: *IEEE Intelligent Systems*, Vol. 16, No. 1.

SURE, Y. – ERDMANN, M. – ANGLE, J. ET AL. (2002). Ontoedit: Collaboratove ontology development for the semantic Web. In: *Proc. 1st Internetional Semantic Web Conference (ISWC2002*), June 9-12th,2002, Sardinia, Italy Springer.

TAN, P. N. – KUMAR, V. – SRIVASTAVA, J. (2000a). Indirect association: mining higher order dependencies in data. In: D.A. Zighed, H.J.Komorowski, J.M.Zytkow (eds.) *Proc.4th European Conf.on Principles and Practice of Knowledge Discovery in Dabases,* Lyon, 13-16 September 2000, Springer, pp. 623-637.

TAN, P. N. – KUMAR, V. (2000b). Interestingness measures for associations patterns: a perspektive. In: *Proc.KDD 2000 Workshop on Postprocessing in Machine Learning Data Mining,* Boston.

TAN, P. N. – KUMAR, V. – SRIVASTAVA, J. (2002). Selecting the right interestingness measure for assotiation patterns. In: D.Hand, D.Keim, R.Ng (eds.) *Proc. 8th. Int. Conf. On Knowledge Discovery and Data Mining,* Edmenton, 23-26 July 2002, ACM Press, pp.32-41.

TOLLE, K. M. – CHEN, H. (2000). Comparing noun phrasing techniques for use with medical digital library tools. In: *Journal of the American Society for Information Science*, Vol. 51, No. 4, pp. 352-370.

VAN HARMELEN, F. – HORROCKS, I. (2000). Questions and answers about OIL, In: *IEEE Intelligent Systems*, Vol. 15, No. 6, pp. 69-72.

ZHONG, N. – LIU, J. – YAO, Y. (Eds.) (2003) *Web Intelligence*. 1st edition. Springer, pp. 440.

WASSERMAN, S. – FAUST, K. (1994). *Social Network Analysis*, Cambridge University Press, New York

WATERHOUSE, S. – DOOLIN, D. – KAN, G. ET AL. (2002). Distributed search in P2P networks. In: *IEEE Internet Computing*, Vol. 6, pp. 68–72.

DIEL II:
VYBRANÉ TÉMY PROGRAMOVÝCH A INFORMAČNÝCH SYSTÉMOV

6 MODELING THE USER IN THE WEB-BASED APPLICATIONS

Information technologies have become a part of our lives not so long ago and especially using of the Web services have become world wide popular. The reason why it is so has caused its availability almost for everyone. The Web is the source of the information for any age groups. But not everybody is computer-literate and able to find the appropriate information effectively. Some people can find what they are looking for very fast, but the others cannot. Even skillful people appreciate if someone or something makes their work easier as well.

The Web environment is very fast and dynamically growing area and the amount of the information that the Web contains grows the same fast. Therefore, it is not unusual that the user gets lost in the information space or is reading the content she does not want to. A way to improve efficiency in information acquisition offers personalized approach concentrated on user's particularities that are employed not only for adapting content but also for the layout or navigation through the space to the individual user.

To fulfill this intent it is important to capture user's particularities, which make the user different for the others and make the base for the personalization. All the user's characteristics are stored in the user model that is the necessary element for the adaptation. The goal is to accomplish the adaptation based on user model as accurate as it is possible. For this purpose we have to concentrate on analyzing the real users of web-based applications and identify the right way how to construct models reflecting them.

The user model changes as the web-based application is used and these changes impact the adaptation. We focus on approaches to the design of the user model, methods for acquiring, representation and employing user model in a way such the user model reflects a real user in web-based application or eventually in more applications. It is necessary to choose an appropriate form of representation of the user model to achieve the expecting personalization.

Throughout this work we present many examples to illuminate defined terms. We have decided to keep consistency by using examples from the only one area. In the examples we suggest a user who is looking for a job and application domain is thus labor market.

6.1 Approaches to user modeling

User modeling is a natural consequence of the society request for the personalization. Approaches to the user modeling employ methods from different areas. The roots of

the user modeling are related to the ITS (Intelligent tutoring systems). ITS consists of three main parts: knowledge about the domain or what to teach, learner specifies who to teach and teacher strategy means how to teach. If we look at ITS from the user modeling view the learner who represents the user of the ITS is the most important part. Katie Hafner defines ITS as an educational software containing an artificial intelligence component. The software observes students' work, tailoring feedback and hints along the way. By collecting information on a particular student's performance, the software can make inferences about strengths and weaknesses, and can suggest additional work (Hafner, 2004).

In Kobsa's work we can find the term user modeling shell systems (Kobsa, 1993, 2001) or abbreviated shell system or shell. This term describes the software tools that provide the representation of assumption about one or more user characteristics of individual user or common characteristics of the user belonging to the group, records of the user's behavior, etc.

In this section we discuss content-based filtering, collaborative filtering, predictive statistical, stereotype and overlay models. All these approaches can be used as stand-alone techniques for constructing user model however combination is possible as well.

6.1.1 Content-based and collaborative filtering

Information filtering aims at reducing a huge amount of the information that the user receives. Special types of the information filtering are content-based and collaborative filtering.

Content-based filtering methods depend on three main factors: items content, ratings given by the user and filtering algorithm (Polčicová, 2004). Content-based filtering can exploit information only from document contents (words, phrases, etc.). This approach evaluates documents content, which the user has found interesting. Algorithms for content-based filtering need the large amount of the information related to the user's behavior.

Collaborative approach or social filtering as it is sometimes called supposes that users with similar taste will likely prefer similar things. Therefore users are divided into groups by their interests. Systems provide recommendations to the particular user based on ratings acquired from other users who had similar preferences in the past. Eliciting explicit feedback (rating about things) from many users is thus necessary to get these likes.

The difference between the content-based and collaborative filtering is that content-based filtering is based on finding the match between content and user's preference whereas collaborative filtering finds the match between people with similar preferences (van Meteren & van Someren, 2000).

6.1.2 Predictive statistical models

The above mentioned approaches to the user modeling serve as a base for predictive statistical models. Statistical models take the advantage from well-investigated area of mathematical statistic and use it for description of existing and prediction of new user characteristics. Overview of the predictive statistical models for user modeling is described more in details and with examples in (Zukerman & Albrecht, 2001). Bellow we offer briefly described main idea of predictive models.

The basic models are:

- linear model,
- TFIDF-based (Term Frequency Inverse Document Frequency) model,
- Markov model,
- neural networks,
- classification,
- rule induction and
- Bayesian network.

Linear model

Linear models derived from regression model use weighted observable and non-random quantities to infer a value for an unknown quantity. Sometimes is the sum of these values extended with the random variable that express errors. Expected value for that variable is 0.

TFIDF-based model

TFIDF expresses the importance of the words to the document. Weighted terms from a document are represented as a vector of weights. Typically the similarity between compared documents is calculated as an angle between vectors. The cosine function is used to evaluate it.

Markov model

Markov model is based on probability distribution. Any state is represented by its probability. Probability of the state is propagated from one state to the other one, i.e. next event depends only on the probability of previously observed events. Special kind of Markov model is Hidden Markov model where states are hidden.

Neural networks

The structure of the networks, non-linear thresholds and the weights of the edges between the nodes make them capable in combination with content-based filtering to represent user's characteristics. Then, the neural network can be learned where the nodes represent the source in that the user is interesting and edges represent strength of the association among sources.

Classification

Classification divides objects described with attributes into the classes. Objects that are in the same class have some common traits or their attributes are close. On the other hand objects with different attributes are located in variant classes.

Rule induction

The method for the rule induction consists of learning sets of the rules. The rules are inducted by using variable techniques upon learning set. Several representations of rules are possible, e.g. direct representation, decision trees, representation in terms of conditional probabilities.

Bayesian networks

Bayesian network is oriented acyclic graph where nodes represent variables and arcs represent dependences among variables. It is necessary to specify for each node the probability distribution that assigns a value to that node taking into account any combinations of the values coming from parents' nodes.

6.1.3 Stereotype model

The mostly used approach in adaptive web-based applications is using stereotype and overlay model or in some cases wise combination exploiting advantages of both models.

A complex research about stereotypes has done Elaine Rich. She defines a stereotype as a collection of frequently occurring characteristics of users (Rich, 1998). Practical results of her research are presented in the system GRUNDY where facets represent user characteristics.

User's knowledge background about the application domain differs from user to user. Stereotype approach takes into account also that aspect. Users are associated with one or more stereotypes that mostly reflect user's knowledge (Kavcic, 2000), social background, computer experience etc. These stereotypes can be ordered hierarchically (Nébel et al., 2003) and characteristics are inherited from superior stereotype. User associated with the stereotype will inherit all stereotype characteristics automatically.

As an example we suppose three user stereotypes related to the user's education: elementary, secondary and higher. The user who has reached higher (university) education is the specialist in the graduated field of science, is able to work independently, invent new solution and bring them into the real life, etc. In the user model we will find all these characteristics even user does not cover any of them. Upon these characteristics we offer higher level job position to the user with university education, for instance positions like project manager or pharmaceutical researcher. On the other hand we do not offer positions like driver or painter that we would rather offer to the user with elementary or secondary education.

Kobsa has identified three important tasks for designers of user modeling components that are needed to be fulfilled (Kobsa, 1993):

- User subgroup identification upon similar characteristics that are relevant for designed system.
- Identification of small number of key characteristics that allows associate a user with the particular subgroup.
- Formalization of the user group characteristics in appropriate representation system is necessary.

Designers play important role in building user stereotypes. They have to identify, which user's characteristics should be included into the stereotype. Kay distinguishes handcrafted and empirically-based approaches to building stereotypes (Kay, 2000a).

Designer builds handcrafted stereotype based on her observation of the group. Empirically-based approach collects data about the users' actions to build stereotype. Designer defines when the stereotype is triggered and how works.

Essential elements of a stereotype are triggers, inferences and retraction facility (Kay, 2000b). Trigger activates the stereotype, i.e. upon information acquired form the user is activated one or more available stereotypes. Seldom can this information be acquired by observing the user for a short period of the time (Kay, 2000a). After assigning the stereotype inferences about the user can be accomplished. User model is thus extended by inferred characteristics. In the case when characteristics obtained in the stereotype do not reflect initial stereotype retraction facility deactivates it.

A different approach to the stereotype user model is a representation where model consists of a set of pairs that assigns value „true“ if user belongs to the stereotype and „false“ in opposite case or it can be a value that reflects probability of belonging to that stereotype as well (Brusilovsky, 1996). Here rises the main drawback of that approach, i.e. the stereotype default assumptions are prone to inaccuracy (Kobsa, 1993). Even if stereotype has been built and applied correctly some inappropriate characteristics have been inferred for particular user. In like manner Kay defines stereotype as a statistically based reasoning about group of people (Kay, 1995).

It is important to emphasize that personalization to the particular user is not possible in this case. Advantage of using stereotypes is visible in initializing user model by first using of the system. The user who has provided some not identifying information (Callaway & Kuflik, 2005) to the system can be associated with the particular stereotype and some characteristics can be inferred.

6.1.4 Overlay model

The main idea is that user's knowledge of the subject is subset of the domain knowledge and the user model is an overlay over the domain model (Brusilovsky, 1996, Kavcic, 2000, Kay, 2000a). The overlay model supposes an instance of itself for any user. This makes a base for personalization for particular user in contrast to stereotypes.

Domain model is in adaptive web-based applications usually represented through concepts and their relationships. Any domain concept has a corresponding value in overlay model in that is stored value represented the user's knowledge of the concept. Estimation of the concept can be binary, qualitative or quantitative value (Brusilovsky, 1996).

An open source general-purpose adaptive hypermedia system AHA! uses the overlay user model. The user model consists of the set of concepts with corresponding value (de Bra et al., 2003). Apart from additional personal concepts stored in the user model the overlay user model in AHA! incorporates any concepts from domain model. An appropriate value is assigned to the concept by the system. This value expresses individual user's knowledge level of this concept.

Special kind of overlay model is differential user model (Kay, 2000a). The differential model does not reflect whole domain knowledge as is represented in a domain model but only part of it. In this part is only subset in which the user might be interested in. Overlay model in that is only real subset of domain knowledge and incorrect knowledge is not included is called strict overlay model (Kavcic, 2000). User model including also incorrect knowledge is called perturbation model (Carmona & Conejo, 2004).

Main drawback of this approach is a necessity of initialization of the model. When system starts for the first time there is no information about user's knowledge for particular domain concept. If system uses quantitative estimation the initial value is set

to 0 or to some mean value, e.g. 50. This approach uses usually educational adaptive web-based applications. The way how to overcome this problem is a combination of overlay model with stereotype model (Wu et al., 2000).

6.1.5 Shared model

Important issue related to effective usage of the user's characteristics representation is sharing the user model among several applications. This brings several advantages as shown in (Bieliková & Kuruc, 2005). System can use the initialized data from the others, what may prevent the user to type the same information again and again in every system. However, the key advantage of the shared user model is an availability of user's characteristics discovered by other system.

Noticeable advantage of shared user model is that information is logically stored at one place (from technical point of view it can be replicated). This eliminates the redundancy and simplifies the maintenance of the model. All changes need to be performed only once. Moreover, when we have available characteristics from various information domains at one place, we can use them to build more complex user characteristics.

Drawback of shared user model lies in its essential lays in the very principle of it. Applications may use different names for the same concepts, which can lead to semantic duplicities in the model. This is the situation, where ontologies are useful – to establish common names of „things“ Another problem arises when the applications using the shared user model evaluate some user's characteristic differently. This characteristic would be constantly changing as the user uses various applications, which can lead to poor results of personalization among all applications, which use the mentioned characteristic. A solution to this problem is to keep track of the model changes. This would allow each application to use this track as additional information for personalization.

Many web-based applications define user model as a single-unit model. In spite of this fact many models have been developed independently although we could find at least couple characteristics, which are common for them. Designing user model as a single-unit might cause reusability problems. Therefore, decomposition of the user model into the parts seems to be in place. Here is necessary to identify characteristics, which should belong to individual parts. A domain dependency described in next section seems to be a properly measure for dividing user's characteristics into groups. This leads to designing a general user model consisting from domain-independent characteristics, which reflect everything what is suitable for most of the users in different web-based applications and the rest of the characteristics.

6.2 User characteristics and sources for user modeling

User model consists of various characteristics. Some authors use for describing the same meaning terms like attributes (Basu et al., 1998), features (Brusilovsky, 2001) or properties (Jameson, 2001). For the purpose of this work we use the term characteristic. Characteristic is a feature that describes the user. Set of the user characteristics creates the user model. Characteristics about the user stored in the user model make a base for the content, navigation and presentation adaptation to the particular user. The more

relevant characteristics describing the user are included in the user model the more accurate is the adaptation.

As an example about how these characteristics can help the adaptive process let's assume that we have a characteristic in the user model expressing the minimal wage per month acceptable by the user. Thus if the system knows that the user is not interested in job offers where offered wage per month is less then her expected wage, it will not present her offers that do not fulfill this condition. System is filtering the information on behalf of the user and the user model assists in this action.

There are several user characteristics classifications. We have identified two main criterions that allow distinguish user characteristics into the groups. On the one hand it is criterion taking into account time and on the other hand it is domain dependency.

6.2.1 User characteristics classification

Brusilovsky identified goals, knowledge, background, hyperspace experience and preferences as the user characteristics (Brusilovsky, 1996). Later he appends user's interests (long-term or short-tem) and individual traits (user's features group in order to define user as an individual) and omits hyperspace experience (Brusilovsky, 2001).

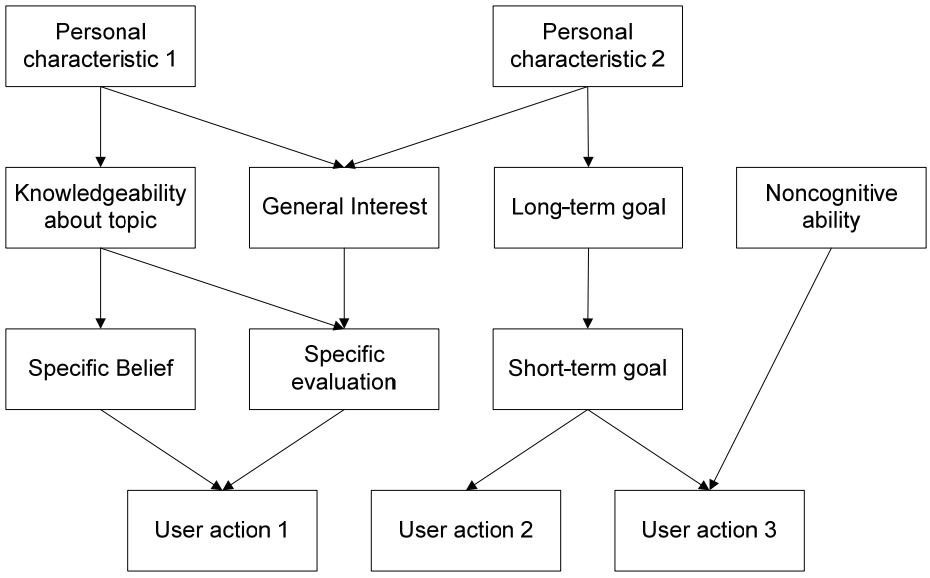


Figure 6-1. Typical relationships among properties (Jameson, 1999).

Jameson uses for all characteristics one general term – property (Jameson, 1999). Under this term are hidden personal characteristics, general interests and preferences, proficiencies, noncognitive abilities, current goal, beliefs, behavioral regularities, psychological states and context of interaction. Typical relationships among properties are shown in the Figure 6-1. The arrows in the pictures mean that the source property influences the target property.

One of the approaches is distinguishing user characteristics in permanent and variable group (Nébel et al., 2003). As permanent characteristics authors consider these that are independent of experience with the particular application. Here belong interests,

attitudes, personality, skills, knowledge, abilities and preferences. Variable characteristics are divided into two subgroups. First subgroup covers characteristics that do not depend on using the application and can be changed independently, e.g. age. The other subgroup covers characteristics that are changed with experience of use of the application, e.g. working with the application raises the user level.

Similar approach taking into account how often are characteristics changed divides them into the static and dynamic group (Kavcic, 2000). The static group covers characteristics that has been set once and are not changed throughout lifetime of the user model or are changed rarely. To this group belongs for instance name, gender, date of birth, etc. Dynamic characteristics are usually related to knowledge, goals, etc. and express part of the user model where changes are made more often, e.g. user's home address. We suppose that people are moving from one place to the other one and merely seldom they live on the same address for whole life. These two groups differ also in the way how the information stored in these characteristics is acquired. The static characteristics (e.g. gender, date of birth, etc.) are mostly asked in some questionnaire whereas the dynamic characteristics are acquired by observing user's behavior throughout working with the application. For instance we can record information about the job offers that has user already visited. Keeping records about visited offers is not aimless. Observing is a good source for future inferring user's preferences or interests.

The domain dependency is an important distinguishing criterion (Kavcic, 2000). This criterion allows detach user characteristics into the parts. Domain dependency criterion divides user's characteristics into domain-dependent and domain-independent group. Domain-dependent characteristics are close connected to the application domain, for instance preferred wage per month or per hour in the case of labor market domain. On the other hand domain-independent characteristics are not related to the application domain, for instance age or gender. Detaching domain-independent part opens the way to build common user model that could be reusable in different applications.

For some domain-dependent characteristics we can use the term preference and interest. The term preference expresses user's expectations towards the application domain. Example of the preference is that the user prefers to work as a human resources manager to technical support specialist. Term interest is not as conscious as the preference. Interest expresses attractiveness about someone or something to the user (González, 2003) and generally is less accurate. In that case user just knows that he is interested in the work with the people and he likes manage them. But she does not know if she wants to work as a human resources manager or consultant.

6.2.2 Sources for the user modeling

In previous sections we have already mentioned that we need much relevant characteristics describing the user to accomplish accurate adaptation. If we want to provide high level personalization we need identify sources from which we can extract information.

Sources for user modeling are mostly based on intuition and/or experience of the system developer (Kobsa, 2001). The prior work and experiences of designers are very invaluable sources for the developing of the user model. Designers use experiences and time-consuming research to identify all the important characteristics that are supposed to be included into the user model. Not a trivial part of this research is the commu-

nication with specialists in the area from the praxis for which the user model is designed to. After the identifying phase we can start with filling in the user model with collected information.

There are two ways how to collect information about the user (Callaway & Kuflik, 2005): explicitly and implicitly. Explicit way is based on asking the user to provide information, implicit way exploits information acquired by monitoring user's behavior and inferring user's characteristics form accomplished observations.

The source of the information coming from the user during her work with an application is called feedback. Here we can distinguish implicit and explicit feedback. The feedback when the user is not disturbing until she is working we consider as an implicit feedback. On the other hand when the user answers questions, or fills in questionnaires we consider as an explicit feedback, because involving the user is necessary here. We discuss user feedback with examples more in details in section 6.5.2 where we point out advantages and disadvantages of using it.

At this point we have to notice that besides filling in forms we can find out information about the user from external sources, e.g. structured documents. An example of this document could be structured curriculum vitae. For this kind of sources we need specialized one-purpose tool, which is able extract information from the document and store them into the user model. The main drawback is that developing general tools for extracting from various sources is not a trivial problem. If we have extracting tool, which is able achieve good results we can consider them also reliable like information acquired from the forms because the source of this information is the user as well.

Observing the user's behavior (Wu et al., 2005) or navigation in the information space during her work with the application is another source. The main advantage of this approach is that the user does not even have to know about it. That way we can find out details about the content, which has the user read, when, what time has he spend there, etc. From this we can infer interesting characteristics about the user. Analysis of the user's behavior is insufficient if it is performed at the end of the session. It is necessary to do some analysis during the session (Ardissono & Torasso, 2000).

For instance we could find out that the user has spent a lot of time reading job offers related to the information technologies and no time reading the offers form different areas. Sometimes we can infer a characteristic, which we would not find out from the user if we asked her explicitly. Reliability in this case is not always guaranteed because we have to rely on inferred information.

6.3 Semantic Web environment

Adaptive hypermedia has brought an important element to the Web environment – personalization. Personalization is helpful for several existing problems on the Web. At least we should mention situations when the user is overloaded by the huge amount of the information, or can get lost in the information space, or is visiting a content she does not want to see, etc. Personalized approach to the user allows her to work more effective with information that is presented.

The Web as we have known it so far has become successful very fast. The reasons why it is so are following (Haustein & Pleumann, 2002):

- simplicity of HTML language,

- immediate feedback after HTML page had been designed,
- additional benefits, because HTML pages are not used only to present information, but e.g. a means of discussion or documentation for people participating in a project,
- low critical mass of people was needed to raise their interest to become involved.

Here is important to notice that even a content presented to the user is personalized it is still suitable only for a human. One has to make sometimes not a small effort to make this information useful. This is significant especially when the user is looking for information. After constructing a query what can be sometimes time consuming and for not computer-literate people very complicated action, the user gets a list, which is assembled only after a text processing (a fulltext comparison between query and data stored in databases). For instance, if user's keywords are job and programmer because we want to find a job position for a programmer, the result is all documents containing these words. These documents do not even have to be job offers. It might be documents describing programmer's job in generally. Here is no guarantee that this list contains links to documents the user has been looking for and the user has to browse them to extract information. This result also does not take into account relations among information that are for a human obvious at a first sight.

A solution to this problem is to represent the Web content not only in a form understandable for a human but also in a form that is easily machine-understandable. In (Antoniou & van Haremelen, 2004) is the term machine-understandable consider as not very appropriate even it is used quite often. This term makes people think that computers can really understand. Authors incline rather to use the term machine-processable instead.

Here comes Tim Berners-Lee's vision called Semantic Web initiative. This initiative tries to add semantics to knowledge to make it processable by automated tools as well as by people.

Probably the easiest way how to capture the meaning of the presented content provides metadata (data about data), which can be added to the presented content. But this solution solves problem related to the machine processing only partially.

In a job offer might be information as follows:

```
<jobOffer>
  <position>Java Programmer</position>
  <company>IT Solution</company>
  <salary>10000</salary>
  <startDate>ASAP</startDate>
  <contractType>Temporary</contractType>
  <region>New York</region>
  <advantage>
    <benefit>car</benefit>
    <benefit>mobile phone</benefit>
    <benefit>health care</benefit>
  </advantage>
</jobOffer>
```

Any author of the web page content use own terms to describe semantics of the presented information.

A movement from current state where information on the Web is in a natural language is not easy. Therefore Tim Berners-Lee proposed a transition to the Semantic Web in steps (Berners-Lee, 2001) and the progress proceed to the next step when the previous is already fulfilled. These steps are known as the „layer cake“ and are shown in the Figure 6-2.

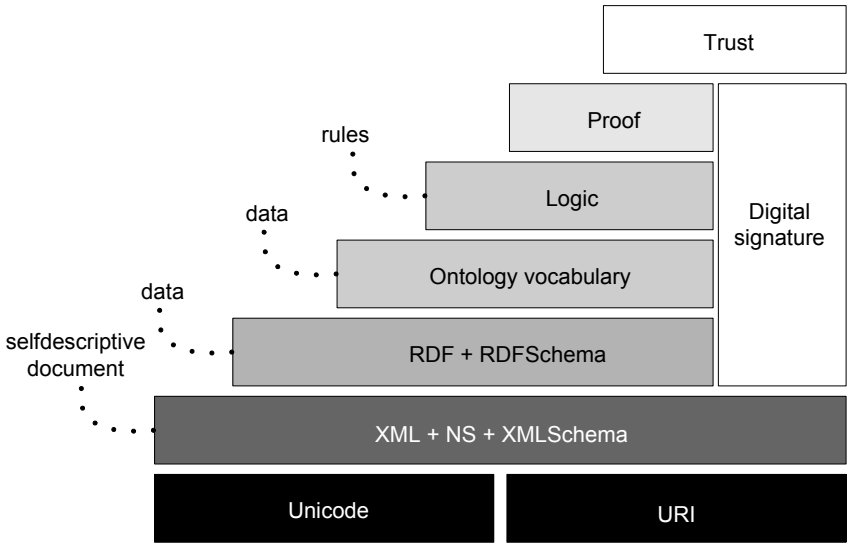


Figure 6-2. The Semantic Web layers.

The two bottom layers (Unicode, URI and XML+NS+XML Schema layer) make a syntax base for the Semantic Web languages. The Unicode layer provides a basic scheme for encoding of the international characters sets. URI is the means for unique identification and addressing of the resources on the Web. The XML layer with namespace and schema definitions helps integrate the Semantic Web definitions with the other XML based standards. RDF allows writing statements about resources and RDF Schema (based on RDF) allows building hierarchies from these resources. But RDF Schema is still not enough powerful and more expressive ontology languages are needed to express more complex relationships. On higher level are ontologies with vocabularies and relationships between concepts what make ontologies an appropriate way how to represent various models in the web-based applications. Therefore, we discuss ontologies and ontology languages more in details in standalone section. The Logic, Proof and Trust layers are still being researched and we do not describe them here because these layers have not been standardized yet.

6.3.1 Representing data by ontology

More complex solution than metadata provides ontology. The term ontology originates form philosophy, namely from the study of the nature of existence. Several ontology definitions have been given so far. For the purpose of this work it is not necessary to mention any of them. We have decided to use Studer's et al. definition, which goes out from Gruber's definition. An ontology is a formal, explicit specification of a shared conceptualization (Studer et al., 1998). Denotation of this definition is following:

- *formal* – ontology should be machine readable,

- *explicit* – type of used concepts and constraints are defined explicitly,
- *shared* – ontology is not private to an individual, but it is accepted by a group,
- *conceptualization* – refers to an abstract simplified view at the world, which is formally represented.

If we take a look at ontology from the subject of the conceptualization point of view, we distinguish following kinds of ontologies (Guarino, 1997, 1998):

- *Top-level ontology*. Describes very general concepts like space, time, matter, object, event, action, etc., which are independent of a particular problem or domain.
- *Domain ontology*. Many times the term ontology is narrowed only to the modeling of the domain. Domain ontology describes the vocabulary related to generic domain (e.g. labor market domain) by specializing the terms introduced in the top-level ontology.
- *Task ontology*. Describes the vocabulary related to the generic task or activity (e.g. looking for a job) by specializing the terms introduced in the top-level ontology.
- *Application ontology*. Describes concepts depending both on a particular domain and task, which are often specializations of the both related ontologies.

The graphical view of the kinds of the ontologies with relationships among them is shown in the Figure 6-3. The arrows represent the specialization relationship.

The question is where to assign the ontology, which would describe a user who is using a web-based application. Considering that the user ontology describes the user as a person (a job applicant allocated in the labor market) and so defines a vocabulary for her, we have to assign this ontology to the domain ontology group. However, a job applicant is looking for a job and this ontology have to reflect this fact as well, what is in a matter of fact an action what makes us think about the user ontology in the task ontology range. At last, in the case of the user ontology, the border between assigning to the domain or task ontology group is very thin, because the user ontology as the task ontology could not exist without the domain ontology defining vocabulary in our case for the labor market.

In the ontology we define concepts like classes (general things) in the many domains of interest, instances (particular things), relationships among those things, properties (and property values) of those things, functions, constrains and rules involving those things (Obrst, 2003). This gives the ontology much powerful expressiveness than the metadata have.

Many ontologies are shared even although it is difficult for various applications to cooperate with them because different vocabularies are used. Differences in terminology can by overcome by mapping among different ontologies. To exploit the full power of the ontology it is necessary to use an appropriate language that is able to express all concepts we have described above.

Following ontology languages are available (McGuinness & van Harmelen, 2004):

- XML provides a surface syntax for structured documents, but imposes no semantic constraints on the meaning of these documents.
- XML Schema is a language for restricting the structure of XML documents and also extends XML with datatypes.

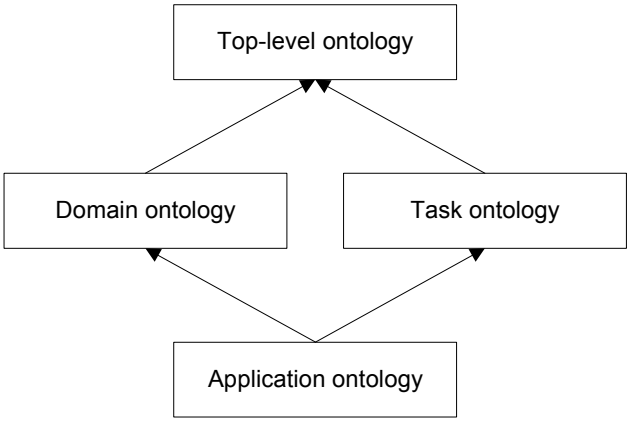


Figure 6-3. The overview of the ontology kinds (Guarino, 1998).

- RDF is a data model for objects („resources“) and relations between them, provides a simple semantics for this data model, and these data models can be represented in an XML syntax.
- RDF Schema is a vocabulary for describing properties and classes of RDF resources, with a semantics for generalization hierarchies of such properties and classes.
- OWL adds more vocabulary for describing properties and classes: among others, relations between classes (e.g. disjointness), cardinality (e.g. „exactly one“), equality, richer typing of properties, characteristics of properties (e.g. symmetry), and enumerated classes.

6.4 User model representation

User model represents various user's characteristics, which can be used for adaptation of the content, presentation or the navigation. Kay and Lum define user model as beliefs about the user that include preferences, knowledge and attributes for a particular domain (Kay & Lum, 2005). At 9th International Conference on User Modeling in 2003 was user model defined as an explicit representation of properties of individual users or user classes. It allows the system to adapt its performance to user needs and preferences (Brusilovsky et al., 2003).

The aim of the user modeling is to identify user's characteristics and chose an appropriate representation of them. Well defined characteristics should reflect a real user of the application as much as possible what is necessary for successful personalization. An appropriate representation with requested expressiveness is also needed. Otherwise is not likely that we achieve desirable adaptation.

There are several approaches how to express and store the user model. In this section we describe possibilities of the user model representation and discuss advantages and disadvantages of these approaches.

6.4.1 Relational database

Probably the most obvious is the use of the relational database to store the data about the user, because most of the information systems already use this kind of storage to store its application data. User model is realized as a set of connected database tables. User characteristics are mapped to the various columns of these tables, whose values express the level of the particular characteristic.

If we take a look at data stored in database from presentation point of view, this data does not need any special preprocessing before they are presented to an end-user. Therefore a transformation to the form suitable for presentation supported by web-based applications (HTML or XHTML languages) is comfortable. Here we consider all the application data and also user model data even this data are not primarily addressed for presentation.

Using relational database is quite straightforward, offers good performance and many other advantages like security, data recovery etc. However, relational databases are not primarily designated to express the knowledge, they can not be easily shared and bring difficulties when some changes need to be done.

6.4.2 XML-based representations

Other frequently used approach is the use of the file system to store the user relevant data. In this case, we can develop our own file format from scratch, but doing it, we block almost totally the sharing and reuse of our model. Better is to represent the model in some XML-based language that offers enough powerful expressiveness.

XML is used as a uniform data exchange format between applications. It provides a surface syntax for structured documents but imposes no semantic constraints on the meaning of these documents (McGuinness & van Harmelen, 2004). The content described by XML is separated from formatting, what allows that the same information can be presented to the end-user in different ways. XML does not have a fixed set of tags but allows users to define tags of their own (Antoniou & van Harmelen, 2004). A richer defining of the structure of the XML documents offers XML Schema. An improvement is not only in readability but it provides better possibility of reuse.

XML based user model is used e.g. in the AHA! The part of the user model, which stores information about the user's name and contact email is defined as follows:

```
<record>
  <key>personal.name</key>
  <value>John Smith</value>
  <firsttimeupdated>false</firsttimeupdated>
</record>
<record>
  <key>personal.email</key>
  <value>john.smith@mail.com</value>
  <firsttimeupdated>false</firsttimeupdated>
</record>
```

The performance of this solution is limited by the performance of the used file system, reusability and sharing is better than in database approach, but is still not ideal.

An extension to the XML structure to be able represent graph structure adds UserML (Heckmann & Krueger, 2003). It is accomplished by connecting several modules via

identifiers (IDs) and references to identifiers (IDREFs). Therefore graph structures can be represented instead of trees. This approach uses two cooperative levels. First one defines simple XML structure for the user model entries and second one are categories defined in the ontology. The advantage of this approach is that different ontologies can be used with the same UserML tools.

The both above mentioned approaches (databases, file system) offer only the flat structured describing of user's characteristics in the user model and do not offer any added value from the user modeling point of view.

6.4.3 Ontology-based representation

A simple list of user's characteristics with assigned values lacks semantics what is significant for previous approaches (Bieliková & Kuruc, 2005). A way to move user modeling from the low level describing of the user's characteristics to the higher level with additional possibilities offers ontology-based approach to the user modeling. Ontologies support reasoning across granularities, provide a common understanding of the domain to facilitate reuse and harmonization of different terminologies (Kay & Lum, 2005). Inference represents the important contribution of the ontology-based models. If we have semantically well-defined user's characteristics, we can use the ontology and its relations, conditions and restrictions to provide the basis for the inferential mechanism to infer more characteristics about the user.

A necessary background to putting user model on higher level in practice provides Semantic Web initiative. This initiative tries to add semantic to knowledge to make it understandable by automated tools as well as by people. For this purpose the World Wide Web consortium has released the Resource Description Framework (RDF) and the OWL Web Ontology Language (OWL) as W3C Recommendations.

OWL is a vocabulary extension of RDF (Resource Description Framework) used to make statements about resources. Its model can be expressed in the XML based language. OWL language has three sub-languages with increasingly expressiveness: OWL-Lite, OWL-DL, OWL-Full, where OWL-Lite is the least expressive language and OWL-Full the most expressive one.

Following source code fragment with user's name and her working experience expresses a part of the user model described in OWL:

```
<rdf:Description rdf:about="#name">
  <rdfs:label xml:lang="en">name</rdfs:label>
  <rdf:type rdf:resource=
    "http://www.w3.org/2002/07/owl#DatatypeProperty"/>
  <rdfs:domain rdf:resource="#User"/>
  <rdfs:range rdf:resource=
     "http://www.w3.org/2001/XMLSchema#string"/>
</rdf:Description>
<rdf:Description rdf:about="#hasExperience">
  <rdfs:label xml:lang="en">has working experience</rdfs:label>
  <rdfs:domain rdf:resource="#User"/>
  <rdfs:range rdf:resource=
    "http://fiit.sk/classification#ExperienceClassification"/>
  <rdf:type rdf:resource=
    "http://www.w3.org/2002/07/owl#ObjectProperty"/>
</rdf:Description>
```

The both properties have a domain and a range specified and a property links individuals from the domain to individuals from the range. We distinguish datatype and object properties. Object properties link an individual to an individual. In this case, `hasExperience` is an object property, what means that an individual from the class `User` is linked to an individual from the `ExperienceClassification`. Datatype properties link an individual to an XML Schema Datatype value or an RDF literal (Horridge et al., 2004). An example of datatype property is the `Name` here and it is defined as a string.

For modeling a user seems OWL-DL language based on Description Logic to be appropriate, because it is a good compromise between expressiveness while retaining computational completeness and decidability. OWL-DL enables the ontology developers to use the sophisticated restrictions and provides them with automated consistency control and reasoning, even if this area is still under heavy research. OWL was designed with the intention of the ontology sharing and reuse, which means that models expressed in this language can be easily shared, combined and integrated into other models. It can be expressed as a graph, so enables domain-specific experts to review and comment the model.

By creating ontology-based user model and deriving it from the domain ontology, we raise the probability that the user's characteristics would be shared among a range of systems of the same domain (especially on the Web as the ontologies are represented by the OWL).

In comparison to XML, XML provides only the syntax for structured documents and does not impose semantics on the meaning of these documents. The OWL goes farther and adds more vocabulary for describing properties and classes: among others, relations between classes, cardinality, equality, richer typing of properties, characteristics of properties, and enumerated classes. The OWL is designed for use by applications that need to process the content of information instead of just presenting information to the human (McGuinness & van Harmelen, 2004).

6.4.4 Example of the user model represented by ontology

Ontology-based approach to the user modeling is not a new idea. Semantically described user model has several advantages (Grimnes, 2003), namely increased accuracy of the model, generalization and easier understanding. Some attempts to model a user have been already accomplished, especially in applications aimed at closed information space, like educational systems.

An enhanced ontology-based user model has been developed in an integrated environment for personalized learning content management, called OntoAIMS developed at Eindhoven University of Technology (Denaux et al., 2004). In this project is the user modeled by using OWL-OLM tool for open user modeling. OWL-OLM uses dialogs to elicit information about the user to overcome cold start problem when the user logs on for the first time. User model is linked to one or more ontologies.

Another example is SHOE[8] project. SHOE (Simple HTML Ontology Extensions) is an XML-compatible knowledge representation language for the Web and has been developed at the University of Maryland and allows web page authors to annotate their

[8] Project SHOE, http://www.cs.umd.edu/projects/plus/SHOE/

web documents with machine-readable knowledge. For the purpose of this project a personal ontology has been developed, which defines elements for describing an individual user together with her interests.

At University of Pittsburg has been developed ADAPT^2 – Advanced Distributed Architecture for Personalized Teaching & Training (Brusilovsky et al., 2005) that provides a general framework for distributed education. This framework employs Ontology Server to user model exchange.

We have proposed a user model in the course of the project called Tools for acquisition, organization and maintenance of knowledge in an environment of heterogeneous information resources (Návrat et al., 2005). The outcome from the project is the web-based information system in domain of labor market (both for people who are looking for a job and companies which are looking for employees). Domain model uses ontology-based representation. Domain ontology (explicit conceptualization of job offers) profits from the advantage of ontology reuse and uses also several ontologies (which domain is independent from chosen labor market application domain) to achieve the desired conceptualization, namely *classification*, *region* and *offer*.

Classification (prefix „c“) defines hierarchies for industrial segments, professions, educational levels, qualifications and various organizations. *Region* (prefix „r“) describes the domain of the regions, countries, languages and currencies that are used in these regions. *Offer* (prefix „ofr“) describes general offer domain. General offer is represented by the class `ofr:Offer`. Any offer has a source and a validity interval. The class `JobOffer` is the key class of the domain ontology. This class represents the standalone job offer.

We have designed the user model according the user dependency criterion, which divides user's characteristics into domain-dependent and domain-independent groups. This criterion has led us to the decision to represent the user model by two sub-ontologies corresponding to mentioned groups of characteristics. Our ontology consists of three parts:

- `DomainDependentUser` – domain-dependent part of the user ontology describes domain-dependent user's characteristics,
- `DomainIndependentUser` – contains domain-independent part of the user ontology that reflects domain-independent user's characteristics,
- `UserModel` – is a part that bonds previous two parts together into the user model.

Domain-dependent part is directly connected to the domain ontology and exploits its sub-ontologies – *classification*, *region* and *offer*. Characteristics stored in this part express user's preferences towards the job offer (see Figure 6-4). In this case we store in the user model information about user's desirable job offer. The user can specify following attributes:

- `ContractType` – user has a chance to choose from these options: contract, non executive, permanent, self employed, temporary.
- `Region` – defines the destination where the looking job should be engaged,
- `Salary` – expected wage,

- `DomainClassification` – defines the domain of user's interests, in which the job position should be engaged (usually the standard study programs at universities).

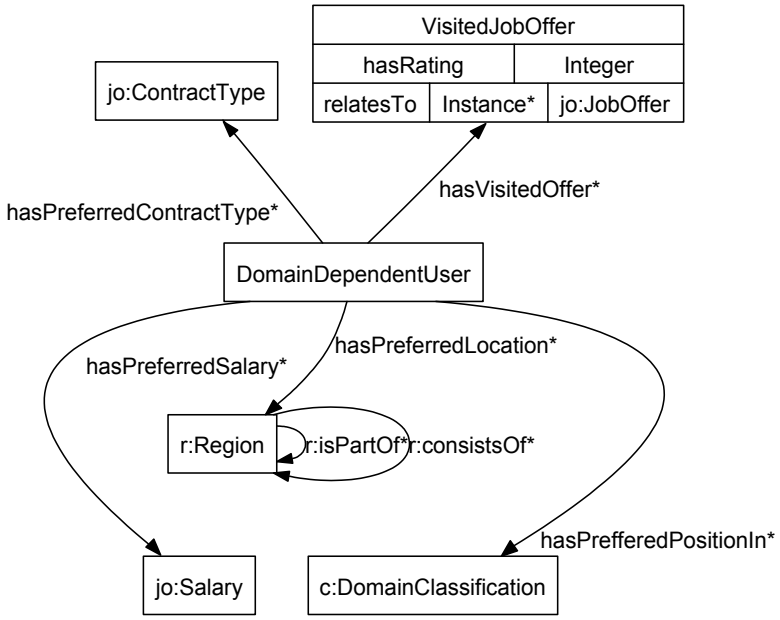


Figure 6-4. Domain-dependent part of the user model.

Information about the job offers that the user has already visited plays important role in the personalization process. This information is modeled by the class `VisitedJobOffer`. The user model stores information about the source of the visited offer (`relatesTo`) evaluation assigned by the user (`hasRating`).

Domain-independent (see Figure 6-5) part involves user's characteristics that describe a user as a person. This part consists from datatype and object properties as well. Datatype properties are `dateOfBirth` and `name`. Object properties are mapped only to the domain ontology parts that are independent from our labor market domain. That way we can express user's language skills including mother language (using reference `hasLanguageKnowledge`), personal characteristics, e.g. gender `hasPersonal Attribute`), computer skills (`hasUserLevel`) and previous education (`hasEducation`).

Proposed user model is derived from the domain ontology, what raises the probability that the user's characteristics can be shared among a range of systems of the same domain. We want to keep extending existing user model to reflect the real user as accurately as needed.

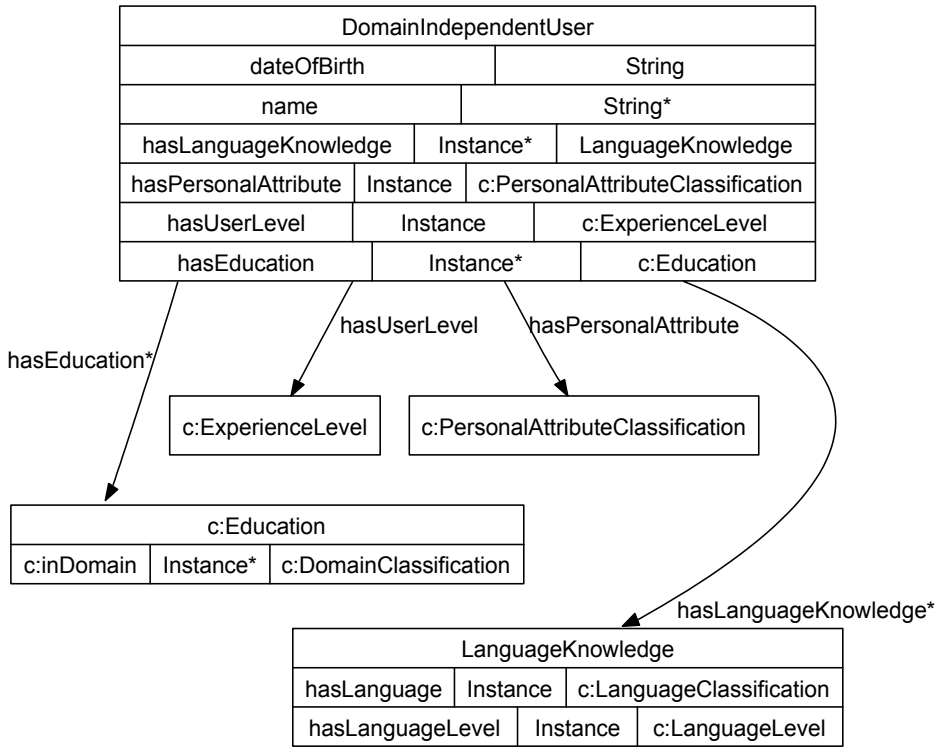


Figure 6-5. Proposal of the user model – the domain-independent part.

When we design general part there is a possibility to use existing ontologies, e.g. FOAF[9] ontology. Although FOAF ontology is not primarily intended for modeling a user in the web-based applications, it contains several personal attributes that might be helpful in designing general user ontology, i.e. first name, last name, email address, etc. Designed user model for any application will then consists of the general part and application specific part, which will differs one application to another. Detaching domain-independent part opens the way to build common user model that could be reusable in applications operating on different domains.

Hence, the proposed user model can be shared across many applications. To this user model base we could add characteristics captured from other applications. But any application uses its own vocabulary for describing user's characteristics. Identifying the same user's characteristics and excluding the redundancy for example in two user models is a trivial problem for a human, but it is not for a computer. Even after flat structured information in attribute-value pair forms where the attribute names are not enough meaningful might be complicated for a human as well. This problem is not so remarkable if for describing in different systems are used synonyms. Ontology-based approach to the user modeling with structured information and additional relations, conditions and restrictions might be the way how to accomplish mapping between two user models automatically.

[9] The Friend of a Friend (FOAF) project, http://www.foaf-project.org/

Besides employing a shared user model there is also possible to use separate user model for any ontology if there are multiple ontologies describing application domain (Brusilovsky et al., 2005). These user models cooperate through Ontology Server, which stores the ontological structures of the domains and also provides platform for information exchange. Here raises a problem when multiple user model servers can work with the same user. It is necessary to employ exchanging mechanisms not to lose important information about the user. It is important to take a look at user modeling from the point of standards to exploit full power of the user model whether it is separate or shared. First attempt to describe user modeling area provides User Modeling Meta-ontology described more in details in (Yudelson et al., 2005).

6.5 Automatic user modeling

In section 6.2.2 we have discussed sources for the user modeling. Several problems are related to these sources. Some of them we have already sketched in that section and some in sections before. Here, we tray to concentrate all these problems in one place, namely we address cold-start problem, user feedback, inaccuracy in the user model, security and privacy in the user model.

6.5.1 Cold-start problem

It is obvious that if we want to accomplish adaptation we need a lot of information about the user. We have identified two main sources of this information. First source is information acquired from the user and second one is acquired by monitoring user's behavior while she is working with the application. Unfortunately, when the user starts using the application for the first time there is no information about her to provide successful personalization. When the personalization is not satisfactory for the user, it can deter the user from future using the application.

We can look at this problem from two points of view – the new-system cold-start and the new-user cold-start problem (Middleton, 2003). The case of the new-system cold-start problem is related to the new system where is no initial information about any user. And the user who uses the application for the first time is the first user since the application has bean launched. In the other case the application has been running for a while and there are records about the other users who have already used this application.

Asking the user questions seems to be a solution. But the user who is asked too many questions usually starts feeling bothered and inclines to check whatever answers just to get through it as fast as it gets. Here, we point out how important is to design the clear and unobtrusive question, but we do not discuss it more in details.

Partial solution gives stereotypes where the user is assigned to the group (stereotype) after he had answered only a few questions. The stereotype reflects characteristics, which are common for most of the group members. If the stereotype is assigned to the user all these characteristics will appear in her user model even any of them does not reflect particular user.

Cold-start problem may be overcome by using default values as well. It is obvious especially in education application. Any of this application is dedicated to a group of students and here we can assume their previously achieved education level. The

knowledge about the student's level of the individual concept was set as an initial value related to the particular characteristic when the user model had been created.

6.5.2 User feedback

Sometimes we need elicit some feedback from the user while she is working. The common example is when the user is asked for an opinion about the content that is being presented. For instance, the user assigns rating to the visited job offer from defined scale. This evaluation could be used for finding similar offers, i.e. if the user likes some offer and assigns it high rating we can suppose that will also like similar ones. This is an example for an explicit feedback. By explicit feedback the user can indicate either positive or negative interaction, e.g. when the user clicks on the button „More details“ indicates a positive interaction or eventually negative interaction by clicking on the button „Not interested“.

When the user's work is not interrupted we are talking about implicit feedback. It requires collecting of evidence and making deduction before it influences the user model. For example, user might read detailed information about a job offer only to find it not interesting after all details are known. Implicit feedback often introduces inference errors to the user model. Explicit feedback has higher priority than implicit feedback because involves the user (Callaway & Kuflik, 2005) however implicit is more preferred because it has little or no impact on the user's normal work activity (Middleton, 2003).

If we take into account web-based applications the user uses a web browser to access the content represented as web pages. During browsing the web pages it happens many times that the user reaches the content, which she does not want to read or just wants to go one or more steps back in the navigation. To accomplish this intent the user presses the Back button and gets to previous page. This gives very important information to the user modeling. From pressing Back button we can infer information about user's interest about the content that is being presented. If the user had spent enough long time before he pressed Back button we can suppose that the presented content is interesting and make changes in the user model.

As a main source for modeling process we consider information collected from the user. Usually there are not too many options how to elicit especially personal information without asking. The mostly used form is questionnaires where the user has several closed questions, i.e. prepared answers are provided and the user only needs to pickup one or more. The amount of the questions and appropriate formulation is very important to make the user do activities, which are not directly related to her goal, i.e. to work with the application. Otherwise the user could be prone to choose answers that do not represent her or choose answers without reading them only to go through very fast. Similarly, sometime the user does not know the correct answer of the questions (e.g. she has not achieved required knowledge yet) or does not want to talk about it because it is too private for them (Rich, 1998).

At last we should not forget psychological aspect of answering questions. People use to see themselves in better light than they in fact are. In this case is all the effort useless. Sometimes is better to work with smaller amount of information than with the large set that is inaccurate. This form of the information acquiring is useful when the user starts using the application for the first time and the user model needs to be initialized. Therefore questionnaires should be unobtrusive as much as possible not to discourage

the user from using the application or provide the answers honestly. This source we thus consider as a very reliable. The main drawback of this approach is that user's active involvement is necessary.

6.5.3 Inaccuracies in the user model

We have identified two sources for the user modeling. As we have already mentioned the information about the user comes directly from the user or is a result of the monitoring of the user's behavior. Information given by the user we consider as the most reliable source.

The problem rises when the information about the user is inferred or is acquired from the monitoring of the user behavior. That way can be inferred not only values that are assigned to the particular characteristics, but also new characteristics. Especially, when the stereotype model is used, the user becomes a member of the group even she does not have any of the characteristics that represents particular stereotype. The main idea of the stereotype model is that user covers most of the characteristics, but only the minority characteristics may cause significant inaccuracy, which will reveal in the unsatisfactory personalization.

Monitoring of the user's behavior and following inferring user's characteristics is interesting from the researcher point of view, but untrustworthy in the user's eyes. Diffidence may cause that user will stop using the application because she does not know what kind of information about her are stored in the user model. To avoid the problem with trust in the user model, the user should have an opportunity to check the content of the user model through some kind of user-friendly interface. Here is important to distinguish and also indicate what characteristic and its value as well is originally acquired from the user and which is inferred.

A special case that may sometimes appear and that causes inaccuracy in the user model is when the user allows another person to work with the application in her name (Kay, 2000b). In this case we will probably notice the unsatisfactory personalization.

6.5.4 Privacy and security

Personalized approach to the web-based applications brings many advantages. But when we provide the user information she wants, we have to keep in mind her security and the right to the privacy. Privacy is defined in the online encyclopedia Wikipedia[10] as the ability of an individual or group to stop information about them from becoming known to people other than those whom they choose to give the information.

Personal demands for privacy in the user modeling can be influenced by such factors as (Schreck, 2000):

- *Personal preferences for privacy in information technology.* It identifies whether the user prefers anonymous or identifiable access to the information.
- *Personal attitudes towards monitoring and classification through applications.* The user can make the decision that she does not agree with the monitoring of her behavior during the work with the application.

[10] Wikipedia, www.wikipedia.org

- *Personal expectations for user adaptive applications and their adaptations.* It means if the added value provided by the system is enough high to make the user feel willing to accept disclosing of her personal information.
- *Personal needs to keep different sets of characteristics of different user adaptive applications apart from each other*. It identifies how large part of the user model (user's characteristics) can be shared among applications.
- *Personal roles, which a user assumes while using a user adaptive application.* Throughout the work with the application there is not only the adaptation the user but also to their roles.

The user model that is used for the personalization is usually stored on the server side together with the application and rest of the application data. It means that user's characteristics, which might be often personal, are stored there as well. But we can not forget that secured connection is also one of the aspects. Therefore, the security in the web-based applications have to refer to the protection against unauthorized access and modification (Joshi et al., 2001), damage, lost or misusage of the information (Kováčik, 2002).

The purpose of this work is not to discuss the security or privacy issues in the Web environment in details. We only point out some aspects of these problems. The main goals of information security are (Kováčik, 2002):

- *Secrecy* – only the authorized user is allowed to use the information.
- *Integrity* – only authorized user is allowed to change stored information and if the opposite case happens, the violation is identified.
- *Availability* – the information is presented to the user at the moment when it is requested.

These three goals might be achieved mutually to provide the maximum security of the user's information in the web-based applications. But in the real applications sometimes happens that achieving the highest level one of these goals causes decreasing another one.

6.6 Lifetime and maintenance of user models

The user model is a part of the web-based applications. It is not stable but changes are made frequently. As the real user is being changed in the real world, the user model does not have to stay back. We can say that the changes are necessary to keep the model up-to-date to reflect the real user.

Considering lifetime we have to take into account characteristics, which are stored in the user model. User characteristics together with their classification are described in section 6.2.1. Hence, we can assume that if the majority of the characteristics belong to the long-term group it is unlikely that lifetime of the user model will be short-term and vice versa. Here we also have to think about how many times the user uses the application. In the application, which is used only once by the particular user is not too much space for adaptive personalization, but there is still a chance for the adaptable personalization. Using the application only once but for enough long time (e.g. couple hours) gives the opportunity for the successful personalization, but we still consider the

user model used in this kind of application as a short-time model. We consider the user model as a long-term usually when the adaptive application is used frequently in a longer term. In this case we have enough time to fully exploit the monitoring of the user behavior and watch changes in the user model.

The user model goes trough several phases during its lifetime. The first step lies on the arms of designers who have to collect the most important characteristics about the real user. After that step had been done programmers chose an appropriate form of representation for the user model and construct it. At this point the model is integrated into the application and its using can start.

When the user starts using the application for the first time the user model is not initialized. This problem is known as a cold-start problem and we describe a way how to deal with this problem in the section 6.5.1. Some web-based applications use default initial values (de Bra et al., 2003). To the user's characteristic is assigned a predefined value that is same for any user but later as the user works with the application the values is updated. As an initial value in the user model we also consider stereotypes that bring an inaccuracy to the user model and have to be corrected (Kay, 2000a)

Once the user model is integrated its maintenance does not stop. During the work with the application some new characteristics might arise or be inferred and this fact has to be taken into account. If any change related to the structure of the model needs the impact of the designers or programmers, the work with the application would be inflexible. Therefore the application has to include several tools or mechanisms, which take care about the maintenance of the model. Tools that monitor the user's behavior and make changes in the user model also belong to this group of tools. Analyzing the user's behavior is insufficient if it is performed at the end of the session. It is necessary to do some of analysis during the session (Ardissono & Torasso, 2000).

The user model can fade in two ways. Many times users just stop using the application and never come back. On the other hand if the user makes the decision to stop using the adaptive application he has the chance to cancel her account. The user model can simply fade and collected information is deleted. This case is not very good for the future user modeling because we are loosing precious data that we could use for other users. But if the user does not want to keep information about her in the application we have to delete them.

References

ANTONIOU, G. & VAN HARMELEN, F. (2004) *A Semantic Web Primer*. Cambridge, Massachusetts: The MIT Press.

ARDISSONO, L. & TORASSO, P. (2000) Dynamic user modeling in a Web store shell. In: *14th European Conference on Artificial Intelligence.* Berlin, pp. 621-625

BASU, C., HIRSH, H. & COHEN, W. (1998) Recommendation as classification: Using social and content-based information in recommendation. In: *Proceedings of the fifteenth national/tenth conference on Artificial intelligence/Innovative applications of artificial intelligence*. Madison, Wisconsin, United States, American Association for Artificial Intelligence.

BERNERS-LEE, T., HENDLER, J. & LASSILA, O. (2001) *The Semantic Web*. Scientific American, 284, pp. 34-43.

BIELIKOVÁ, M. & KURUC, J. (2005) Sharing User Models for Adaptive Hypermedia Applications. In: *ISDA 2005*, Wroclaw, Poland, ACM Press, pp. 506-511.

BRUSILOVSKY, P. (1996) Methods and techniques of adaptive hypermedia. *User Modeling and User-Adapted Interaction*, Vol. 6, pp. 87-129.

BRUSILOVSKY, P. (2001) Adaptive Hypermedia. *User Modeling and User-Adapted Interaction*, Vol. 11, pp. 87-110.

BRUSILOVSKY, P., CORBETT, A. & DE ROSIS, F. (2003) User Modeling 2003: Preface. In: *Brusilovsky, P., Corbett, A. & De Rosis, F. (Eds.) 9th International Conference on User Modelling*, Johnstown, USA, Vol. 2702, Lecture Notes in Computer Science.

BRUSILOVSKY, P., SOSNOVSKY, S. & YUDELSON, M. (2005) Ontology-based framework for user model interoperability in distributed learning environments. In: *Richards, G. (Ed.) World Conference on E-Learning, E-Learn 2005,* Vancouver, Canada, AACE, pp. 2851-2855.

CALLAWAY, C. & KUFLIK, T. (2005) Using a Domain Ontology to Mediate between a User Model and Domain Applications. In: *Brusilovsky, P., Callaway, C. & Nürnberger, A. (Eds.) Workshop on New Technologies for Personalized Information Access*. Edinburgh, Scotland, UK, pp. 13-22.

CARMONA, C. & CONEJO, R. (2004) A Learner Model in a Distributed Environment. In: *De Bra, P. & Nejdl, W. (Eds.) Adaptive Hypermedia and Adaptive Web-Based Systems: Third International Conference, AH 2004,* Vol. 3137, Eindhoven, The Netherlands, Lecture Notes in Computer Science, pp. 353-359.

DE BRA, P., AERTS, A., LANGE, B. D., ROUSSEAU, B., SANTIC, T., SMITH, S. & STASH, N. (2003) AHA! The adaptive hypermedia architecture. In: *ACM Conf. on Hypertext and Hypermedia*, Nottingham, UK, pp. 81-84.

DENAUX, R., DIMITROVA, V. & AROYO, L. (2004) Interactive Ontology-Based User Modeling for Personalized Learning Content Management. In: *AH 2004: Workshop Proceedings Part II,* Eindhoven, Netherlands, pp. 338-347.

GONZÁLEZ, G. (2003) Towards Smart User Models for Open Environments. Department of Electronics, Computer Science and Automatic Control. PhD. thesis. University of Girona.

GRIMNES, G. A. (2003) Learning Knowledge Rich User Models from the Semantic Web. In: *User Modeling, 2003*, Lecture Notes in Computer Science, Johnstown, Pennsylvania, USA, pp. 414-416.

GUARINO, N. (1997) Uunderstanding, Building, and Using Ontologies: A commentary to "Using Explicit Ontologies in KBS Development", by van Heijst, Schreiber, and Wielinga. In: *International Journal of Human and Computer Studies*, Vol. 46, pp. 293-310.

GUARINO, N. (1998) Formal Ontology and Information Systems. In: *Guarino, N. (Ed.) FOIS'98*, Trento, Italy, IOS Press, Amsterdam, pp. 3-15.

HAFNER, K. (2004) *Software Tutors Offer Help and Customized Hints.* NYTimes. [Online; accessed February 26th, 2006]

URL: http://www.aaai.org/AITopics/html/tutor.html

HAUSTEIN, S. & PLEUMANN, J. (2002) Easing Participation in the Semantic Web. In: *International Workshop on the Semantic Web 2002*, Hawaii.

HECKMANN, D. & KRUEGER, A. (2003) A User Modeling Markup Language (UserML) for Ubiquitous Computing. In: *Brusilovsky P., Corbett, A. & de Rosis, F. (eds.), 9th International Conference on User Modelling*, Vol. 2702, Lecture Notes in Computer Science, Johnstown, USA, pp. 393-397.

HORRIDGE, M., KNUBLAUCH, H., RECTOR, A., STEVENS, R. & WROE, C. (2004) *A Practical Guide To Building OWL Ontologies Using The Protégé-OWL Plugin and CO-ODE Tools Edition 1.0.* The University Of Manchester. [Online; accessed February 26th, 2006]

URL: http://www.co-ode.org/resources/tutorials/ProtegeOWLTutorial.pdf

JAMESON, A. (1999) User-Adaptive Systems: An Integrative Overview. In: *UM99*. Banff Centre, Banff, Canada.

JAMESON, A. (2001) Modelling both the Context and the User. *Personal Ubiquitous Computing*, Vol. 5, pp. 29-33.

JOSHI, J. B. D., AREF, W. G., GHAFOOR, A. & SPAFFORD, E. H. (2001) Security models for Web-based applications. *Communications of the ACM*, Vol. 44, pp. 38-44.

KAVCIC, A. (2000) The role of user models in adaptive hypermedia systems. In: *10th Mediterranean Electrotechnical Conference MEleCon 2000*. Lemesos, Cyprus.

KAY, J. (1995) The um toolkit for cooperative user modelling. *User Modelling and User-Adapted Interaction*, Vol. 4, pp. 149-196.

KAY, J. (2000a) Stereotypes, Student Models and Scrutability. *Lecture Notes in Computer Science*, Vol. 1839, pp. 19-30.

KAY, J. (2000b) User modeling for adaptation. In: *Stephanidis, C. (ed.) User Interfaces for All*. Florence, Italy, pp. 271-294.

KAY, J. & LUM, A. (2005) Ontology-based User Modeling for the Semantic Web. In: *10th International Conference on User Modeling (UM'05) Workshop 8*. Edinburgh, Scotland, pp. 11-19.

KOBSA, A. (1993) User Modeling: Recent Work, Prospects and Hazards. In: *Schneider-Hufschmidt, M., Kühme, T. & Malinowski, U. (eds.) Adaptive user interfaces: Principles and practice.* Amsterdam, North-Holland, pp. 111-128.

KOBSA, A. (2001) Generic User Modeling Systems. *User Modeling and User-Adapted Interaction*, Vol. 11, pp. 49-63.

KOVÁČIK, J. (2002) *Information Security*. [Online; accessed February 26th, 2006]

URL: http://user.edi.fmph.uniba.sk/winczer/SocialneAspekty/Kovacik.htm

MCGUINNESS, D. L. & HARMELEN, F. V. (2004) *OWL Web Ontology Language: W3C Recommendation 10 February 2004.* [Online; accessed February 26th, 2006]

URL: http://www.w3.org/TR/owl-features/

MIDDLETON, S. E. (2003) Capturing knowledge of user preferences with recomender systems. PhD thesis. University of Southampton.

NÁVRAT, P., BIELIKOVÁ, M. & ROZINAJOVÁ, V. (2005) Methods and Tools for Acquiring and Presenting Information and Knowledge in the Web. In: *International Conference on Computer Systems and Technologies – CompSysTech' 2005*, Varna, Bulgaria.

NÉBEL, I.-T., SMITH, B. & PASCHKE, R. (2003) A User Profiling Component with the aid of User Ontologies. In: *Workshop Learning, Teaching*, Knowledge, Adaptivity. Karlsruhe.

OBRST, L. (2003) Ontologies for semantically interoperable systems. In: *Proceedings of the twelfth international conference on Information and knowledge management.* New Orleans, LA, USA, ACM Press, pp. 366-369.

POLČICOVÁ, G. (2004) Topographic Organization of User Preference Patterns in Collaborative Filtering. Faculty of Informatics and Information Technologies. Bratislava, PhD thesis. Slovak University of Technology.

RICH, E. (1998) User modeling via stereotypes. In: *Readings in intelligent user interfaces*. Morgan Kaufmann Publishers Inc., pp. 329-342.

SCHRECK, J. (2000) Security and Privacy in User Modeling. Dept. of mathematics and computer Science. PhD thesis. Essen, University of Essen.

STUDER, R., BENJAMINS, V. R. & FENSEL, D. (1998) Knowledge engineering: principles and methods. In: Data Knowl. Eng., Vol. 25, pp. 161-197.

VAN METEREN, R. & VAN SOMEREN, M. (2000) Using Content-Based Filtering for Recommendation. In: *Potamias, G., Moustakis, V. & van Someren, M. (eds.) ECML/MLNET Workshop on Machine Learning and the New Information Age.* Barcelona, Spain, pp. 47-56.

WU, H., HOUBEN, G.-J. & DE BRA, P. (2000) Supporting User Adaptation in Adaptive Hypermedia Applications. In: *On-line Conference and Informatiewetenschap 2000.* De Doelen, Rotterdam.

YUDELSON, M., GAVRILOVA, T. & BRUSILOVSKY, P. (2005) Towards User Modeling Meta-ontology. In: *Ardissono, L., Brna, P. & Mitrovic, A. (Eds.) User Modeling 2005: 10th International Conference*, UM 2005. Edinburgh, Scotland, UK, Lecture Notes in Computer Science, pp. 448-452.

ZUKERMAN, I. & ALBRECHT, D. W. (2001) Predictive statistical models for user modeling. *User Modeling and User-Adapted Interaction*, Vol. 11, pp. 5-18.

7 PARALLEL AND DISTRIBUTED SIMULATION OF DISCRETE EVENT SYSTEMS

Simulation and analysis are two alternative methods for studying the dynamics of systems. There are situations in which analytical solutions can be more accurate and more easily obtained then simulation results. However, many systems cannot be adequately analyzed by standard mathematical techniques. Simulation, on the other hand, is very flexible. In simulation modeling, the system can usually be represented at an arbitrary level of detail. This makes simulation a very powerful tool for studying the dynamics of systems.

A major disadvantage of simulation is its expense. As the range and/or the complexity of systems under study increase, running a simulation in a sequential manner (on one central processor) comes into trouble: it requires an enormous amount of time and/or memory for storing the state of the model. Thus, it is natural to attempt to exploit the power of parallel and distributed computing to accelerate the simulation computations, or to extend the amount of available memory.

In the past two decades, there has been a great deal of interest in parallel or distributed execution of simulation models. The main reason of this is the pragmatic one mentioned above – to increase the performance of simulation of large and complex systems. Parallel simulation is also interesting from an academic point of view, because it represents a problem domain that often contains substantial amounts of parallelism, but is surprisingly difficult to parallelize efficiently in practice (Fujimoto, 1990a). Moreover, simulation tasks form a suitable experimental basis for the study of some issues of parallel or distributed computation, such as performance, synchronization algorithms, parallelization and decomposition, etc.

Similar issues arise in another situation, namely when there is a need for cooperation of multiple simulators which are naturally distributed. The natural distribution may ensue from geographical distribution of the simulators, or simply from their independent development. Deployment of such simulators in one common task requires their cooperation and coordination called interoperability. Here, one of the major issues is the synchronization, similarly to the case of parallelizing a single simulation.

The research area called *Parallel Discrete Event Simulation (PDES)* studies the issues connected to simulation involving multiple cooperating concurrent processes, including both of the above mentioned cases. A huge amount of problems has been solved in the past years, many are still open. There is an annual Workshop on Principles of Advanced and Distributed Simulation (PADS), formerly called Workshop on Parallel and Distributed Simulation, devoted to the area of PDES. The results presented on this workshop document the vitality and the importance of the area.

This document gives a survey of the area of PDES, including issues that have been most intensively discussed in the large amount of literature concerning the area. Some topics, such as the conservative methods and the lookahead are elaborated in more detail, since these are the central research interests of the author.

7.1 Discrete event simulation

Simulation is a tool for studying systems by means of using a model of the system. The model is an abstraction of the system under study containing features which are substantial for the objective of the study[11] (Zeigler et al., 2000). The purpose of simulation is to mimic the behavior of the modeled system using the model. In this document, we are interested in *dynamic systems* where time plays an essential role. Passage of time in the modeled system is captured in the model by the notion of *simulation time* (Neuschl et al., 1988, Zeigler et al., 2000). Using this notion, we can re-formulate the purpose of the simulation as being a way to generate the behavior of the model within some (finite) interval of simulation time $\langle 0, T_{END} \rangle$, denoted as *simulation period* $\mathcal{T}$.

7.1.1 Model state

A substantial ingredient of a model description is a set of *descriptive variables*. The values of the variables reflect the state of the modeled system. The values change as the state of the model changes over time. From these variables a subset can be selected which unambiguously determines the state of the system (Zeigler et al., 2000). The selection forms a tuple of *state variables*, let us denote it $A=(\alpha_1,\alpha_2,...,\alpha_k)$. There can be several alternative selections of state variables from the set of descriptive variables. However, the selected state variables must provide an unambiguous state description. Each variable α_i takes on values from a defined set of possible values $RANGE(\alpha_i)$. Then the state of the model at any time is a point in the *state space* $\mathcal{S} \subseteq RANGE(\alpha_1)\times RANGE(\alpha_2)\times...\times RANGE(\alpha_k)$.

The primary purpose of any simulation is to compute the point in the state space $\mathcal{S}$ for every instant of simulation time within the period $\mathcal{T}$. From this viewpoint, simulation is a means for the *state trajectory* construction (Zeigler et al., 2000). The simulation output of interest usually takes the form of some summary statistics calculated from the state trajectory. These statistics may be calculated in the course of the simulation, even without recording the state trajectory itself. Nevertheless, construction of the state trajectory remains the primary purpose. As mentioned earlier, the time domain of simulations of our interest is a time interval $T = \langle 0, T_{END} \rangle$, i.e. time is continuous in

[11] We will tackle the issues related to modeling itself later in section 7.5.2.

these simulations. As for the state domain, we restrict ourselves to simulations where the state changes occur instantaneously at discrete points in time. We do not consider continuous systems, where the state can change continuously with respect to time. Discrete state changes imply that the state trajectory of the model is piecewise-constant. An instantaneous state change is called an *event*. The event has zero duration. The time instants of *event occurrences* are the points on the time axis which divide the time domain into adjacent intervals. The state is constant during each of these intervals. Systems which can be reasonably simulated in this manner are commonly called *discrete event systems*. Examples of such systems are queuing systems, digital logic circuits, computer networks, traffic systems, etc.

7.1.2 Conceptual frameworks

In order to facilitate the modeling of discrete event systems, some standard ways for describing model behavior have been developed. These are denoted as *world views*, or *conceptual frameworks*. According to (Balci, 1988), a conceptual framework is "a structure of concepts and views under which the simulationist is guided for the development of a simulation model". The most common of them are *event scheduling*, *activity scanning*, and *process interaction* world views (Balci, 1988, Neuschl et al., 1988, Page & Nance, 1994).

The event scheduling world view focuses on events and their effects on state changes. The state trajectory computed in a simulation can be described as a sequence of couples (state, its duration). Such description can be transformed to an equivalent one consisting of sequence of couples (event, occurrence time). Events in the sequence differing only in occurrence time as said to be of the same *event type*. The model description in the event oriented world view contains an *event handler* (*event routine*) for each event type. The event handler specifies state changes and scheduling of other events which should take place as a result of the event occurrence.

The activity scanning world view considers activities as atomic units of operation of the model. Activities have non-zero duration in terms of simulation time. Examples of activities are servicing a customer in a server, or transferring a certain amount of data over a network. There can be multiple concurrent, time-overlapping activities in a model. Activity scanning is similar to the rule-based programming where a rule is specified upon the satisfaction of which a pre-determined set of operations is performed. Each activity consists of two parts:

- a condition which must be satisfied in order for an activity to take place
- operations which are performed upon the satisfaction of the associated condition

To relate it to the event scheduling world view, an activity can be considered as corresponding to two events, one of them occurring when the activity starts, and the other when the activity terminates.

In the process interaction world view, consecutive activities which are logically connected to each other, are encapsulated to form a *process*. A process can be viewed as a sequence of activities. Such notion of the process is in consonance with the common usage of the term. It was introduced by D. E. Knuth within his simulation language called SOL (Knuth & McNeley, 1964). A process describes the life cycle of an object involved in the simulation. In the course of the simulation, the process is responsible for managing a portion of state corresponding to the object, and can

interact with other processes. The advantage of this approach is that the code responsible for managing an object's state is encapsulated in a single process rather than being scattered among multiple events or activities as in the other world views. On the other hand, process oriented simulators require a more sophisticated simulation executive.

It has been shown in (Šafařík, 1983) that all the three world views are equivalent in the sense that any model description in one world view can be translated into any of the remaining two world views.

7.1.3 Event-based simulators

The majority of work in the area of PDES is based on the event scheduling world view. Therefore we will elaborate it in more detail now. The model description focuses on the events themselves. It contains an event handler for each event type. The event handler is a procedure describing state changes associated with the event, and causal relationships of the event to other events. The state changes are expressed by commands updating the state variables, and the relations to other events are prescribed via scheduling of future events.

Having such model description, an arbitrary long state trajectory can be computed using a data structure for storing future pending events called *event list* (*calendar*), and the algorithm shown in Figure 7-1. The event list contains an *event record* for each pending event occurrence. The event record consists of event type, occurrence time, and possibly other information. The occurrence time, or the timestamp of an event *e* is denoted as *ts*(*e*).

```
Initialization ();
while( ! terminationCondition() )
{
     e = cause();        //remove earliest event from event list
     Tsim = ts(e);       //timestamp of the earliest event
     executeEvent(e);    //process the event
}
```

Figure 7-1. The basic simulation algorithm.

The algorithm repeatedly removes the event with the smallest occurrence time from the event list, advances the simulation time to that occurrence time, and executes the corresponding event handler. This approach is called *event driven*, since the simulation time advancement is based on the occurrence time of the pending events.

For the sake of completeness, let us mention that there is also another algorithm which can be used to generate the state trajectory, known as *time stepped* simulation. Here, the simulation time is advanced by small constant steps (increments). The simulation algorithm repeatedly advances the simulation time by one step and simulates all pending events which fall into the current step. This method has some drawbacks when used with discrete event systems, thus we will not consider it any further in this survey.

As can be now seen, a simulation program computing a state trajectory consists of code which can be divided into two parts, as follows:

- The problem independent part containing the simulation loop, the event list data structure and the associated operations on it, random generators in the case of a stochastic model, etc. This part is called simulation processor (Meňhart, 1995), or simulation executive (Fujimoto, 2000).
- The problem dependent (application specific) part containing event handlers, simulation initialization, termination condition, etc. This part is referred to as simulation model (Meňhart, 1995), or simulation application (Fujimoto, 2000).

The simulation processor together with the simulation model form the *simulator*.

7.1.4 Event graphs

For the description of event-oriented simulation models in this document, we use the formalism of *event graphs* (Yücesan & Schruben, 1992). As already mentioned, two fundamental components of a discrete event model are a set of state variables, and a set of events. The model prescribes how the state described by the set of state variables evolves in time. The state trajectory in discrete event models is piecewise constant and events represent the points in time where the state changes.

This interaction between events and state variables is nicely captured in the concept of event graphs. An event graph is a directed graph with events (more precisely event types) as vertices and the edges representing causal relationships among the events. A module of code is associated with each vertex. Execution of the code simulates a state transition caused by the corresponding event.

The basic event graph construct is depicted in Figure 7-2. It is interpreted as follows: the occurrence of event *A* causes event *B* to be scheduled after a time delay *t*, providing condition (*i*) is true (after the state transition for event *A* has been done). If *t* is zero, it can be omitted. Similarly, if the occurrence of event *A* always schedules event *B*, the condition can be omitted and the edge becomes unconditional.

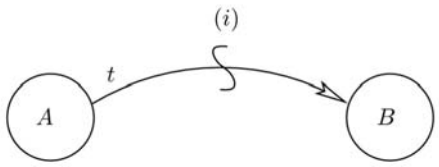


Figure 7-2. Basic event graph construct..

Event graphs have sufficient power to represent any discrete event system (Yücesan & Schruben, 1992). However, some enhancements can increase their readability and ease of construction. One of them are the canceling edges, which allow cancellation of events already scheduled. They are drawn as dashed lines. Another enhancement are parameters passed along scheduling edges. In this way, values can be passed from one event to other events. Consider an element of a graph shown in Figure 7-3. The parameter *pa* on the edge is an expression which is evaluated during execution of the source event *A*. The value is assigned to the "formal" parameter *pf* and can be used during processing of the target event *B*. Technically, it can be thought of that the value of *pa* is stored as part of the event record for event *B*, which can be used when *B* is processed. The other parts of the example are the same as in the previous figure.

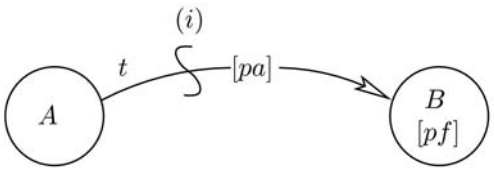


Figure 7-3. Parameters in event graph.

The basic construct of event graphs allows us to introduce another terms. The delay of an edge will be referred to as *scheduling delay*. The *scheduling delay base* is the simulation time serving as the reference point, relative to which the scheduling delay is expressed. Normally, this is the timestamp of the source event. The timestamp of the target event is then calculated as the sum of the scheduling delay base plus the scheduling delay. Another important term related to simulation models is the *operation set* of an event *e*, which refers to the set of state variables the event *e* reads or modifies.

As an example of event graph, let us consider the construction of event graph for a simple queuing system consisting of a single server and a queue (see Figure 7-4).

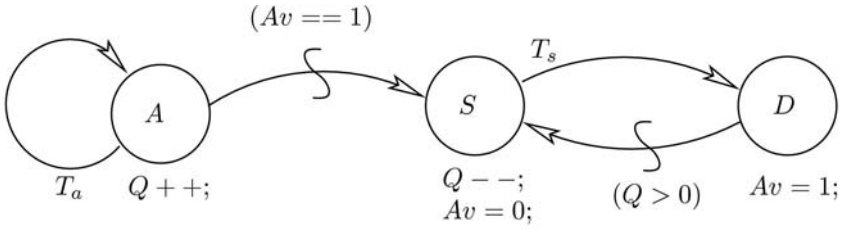


Figure 7-4. Event graph of a simple queuing system.

State variable *Q* represents the queue length, and *Av* denotes the availability of the server. Customers arrive each *Ta* time units, as denoted by the loop adjacent to the *A* (Arrival) event. Upon arrival, the customer is inserted into the queue (*Q* is incremented). If the server is free (the condition *Av*==1), the customer can immediately enter the server, i.e. the *S* (Start of service) event is scheduled with zero delay. This event simulates the customer's progress from the queue to the server, which is reflected by the state variables update. After *Ts* time units, the customer will be serviced and departs, thus the server becomes free, i.e. *Av*=1 in the *D* (Departure) event. Now, if any customers are waiting in the queue, one of them can enter the server, thus another occurrence of *S* event is scheduled immediately. The values of *Ta* and *Ts* can be deterministic, or, for stochastic queuing models, they can be sampled from random generators.

7.2 Parallelizing discrete event simulation

Discrete event simulation controlled by the algorithm mentioned above is *sequential*, in the sense that the algorithm is supposed to be executed by a single processor. In order to accelerate a discrete event simulation, the computational power of multiple

processors can be exploited in several different ways (Fujimoto, 2000). These "model distributions" can be characterized by the kind of computation each of the processors is executing.

In the *parallel replicated simulation*, each processor executes an independent sequential simulation called *replication* (Fujimoto, 2000). This is of benefit if simulation results for a large number of parameter settings are needed, where the results of one replication are not used to determine the settings of another replication. The main advantage of this approach is its simplicity, and no inter-processor communication overhead. A disadvantage is that each processor must hold the entire state of the simulation. This is the only method where multiple replications run concurrently. The remaining methods parallelize a single replication.

The *support function parallelism* is a method where each processor executes a dedicated task of a sequential simulation program, such as event list manipulation, random number generation, and statistics collection. Although this approach can significantly improve performance in some particular cases (Comfort et al., 1984), it offers only limited parallelism and does not scale to large models.

The *time-parallel simulation* partitions the simulation model in the time domain (Lin & Lazowska, 1991, Fujimoto, 2000). The simulation period is divided into adjacent sub-intervals. A processor is assigned to simulate each of the sub-intervals. At a glance, such approach is attractive because of its potentially unbounded parallelism. However, the key issue is that the final state of i^{th} sub-interval must match the initial state of the $i+1^{st}$ sub-interval. One known solution is to "guess" the initial states and perform a "fix-up" if the match has not been achieved.

The other method is to pre-compute the initial states at specific points in simulation time so that the final states will match them. Neither of these methods is generally applicable. They are usable only for specific applications where the fix-up cost, or the state pre-computation cost can be kept low. Such applications include simulation of network medium access protocols CSMA and slotted Aloha (Jones & Das, 2001), trace-driven cache simulations (Nicol et al., 1994), and ATM multiplexer simulation (Andradottir & Ott, 1995). For some other applications e.g. queuing networks, interesting *approximate* simulation results can be obtained using the time division approach. In this case just a portion of the state is compared at the sub-interval boundaries (Wang & Abrams, 1993, Solčány & Šafařík, 1998).

An interesting method which can be considered as a time parallel simulation is based on the *parallel prefix* computation. If the state of the model can be formulated as a linear recurrence equation, a parallel prefix computation can be used to solve the equation over the simulation time (Greenberg et al., 1991). Similarly to the previous, this approach is usable only for specific applications. Moreover, it requires different model description than the usual one based on event handlers.

The most widely researched approach to model parallelization is the *space-parallel* simulation. Here, the modeled system is viewed as a collection of interacting *physical processes*, and the simulation model is partitioned into a corresponding collection of *logical processes* or *submodels*. Each submodel is responsible for managing a portion of state variables and the associated event types. The submodels are assigned to processors. Each submodel is processed by its *local simulation processor* which maintains the *local event list* and *local simulation time*. Basically, the local simulation processor executes the algorithm used to control the sequential simulation (see Figure 7-1). If in

an event handler a need arises to schedule another event which belongs to a different submodel, a message is sent between these submodels. The message contains information about the scheduled event itself and its timestamp, similarly to the event record. Therefore, in this context, the terms "event scheduling" and "message passing" are used interchangeably in PDES literature, and in this document too.

This approach to model parallelization can be thought of as general, i.e. applicable to any system. Another advantage is that each processor holds just a portion of the whole state. The upper limit of the available parallelism is given by the model structure. This upper limit scales with the model size. However, the true parallelism and its scalability achieved in real applications are usually significantly smaller than this upper limit. In the remainder of this survey, we consider the space-parallel approach.

For our enumeration of model parallelization methods to be exhaustive, let us mention the approach devised by Chandy and Sherman (Chandy & Sherman, 1989b). They provide a high level unifying view of simulation parallelization called "space-time" approach. The *space-time* is the cartesian product of the discrete state space $\mathcal{S}$ and the continuous time period $\mathcal{T}$. The objective of the simulation is then to "fill in" the space-time $\mathcal{S}\times\mathcal{T}$.

The parallelization into submodels relies in dividing the space-time into regions of arbitrary shape. The simulation of such model uses an iterative *relaxation* method. A submodel is responsible for filling in the space-time region assigned to it. In order to accomplish this task, the submodel must be aware of boundary conditions for its region, and update them in accordance with activities in its own region. Changes of boundary conditions are transmitted among submodels in the form of messages. Thus each submodel repeatedly computes its portion of the space-time, transmits changes in the boundary conditions to the submodels responsible for neighboring regions, and then waits for new incoming messages. The computation proceeds until no more changes of boundary condition occur.

7.3 The issue of synchronization

Consider a space-parallel simulation model consisting of submodels, running on multiple processors. It is hardly feasible to assume anything about the relative speeds of the processors, and about the load of physical processors. The load of a processor depends on the number of submodels mapped to that processor, the "density of events" on the time axis in different submodels which may vary during the simulation, the computational requirements of individual event handlers, etc. For a while, assume that the submodels are simulated independently, without coordination of the advancement of their local simulation times. Then it can easily happen that a submodel receives a message with timestamp less than its local simulation time. This situation is undesirable, because an event with smaller timestamp has the potential to change the state and thus affect events that occur later. Such violation of causality is called *causality error*.

In sequential simulation, the single event list is a means to establish a total order of events and thereby avoid such errors. In the parallel case, the sufficient condition to avoid causality errors is that events within each submodel are processed in correct i.e. non-decreasing timestamp order. This condition is known in PDES literature as the

local causality constraint[12] (Fujimoto, 1990a). It results in partial ordering of events, similarly to Lamport's notion "happens before" (Lamport, 1978). To adhere to this constraint, the submodels cannot run independently, they must be *synchronized*. Submodel communication and synchronization is performed by the *parallel/distributed simulation processor* using some synchronization method. Synchronization is the key issue in PDES. Most of the research in the area of PDES has been devoted to this issue.

Many synchronization methods (also referred to as synchronization algorithms, mechanisms, schemes, or protocols) have been proposed, with different applicability and efficiency. There are good surveys of the methods, e.g. (Fujimoto, 1990a, Low et al., 1999). Introducing all known methods would require too much space. Therefore, in this section we summarize those which have had greatest impact in the context of PDES, and those which are related to our particular area of interest. The methods can be broadly classified as either *conservative* or *optimistic*. While the former strictly avoid causality errors, the latter provide support for detecting and correcting them. We review methods from both categories and give their comparison thereafter.

7.3.1 Asynchronous conservative methods

In order to prevent causality errors, conservative methods control local simulation time advancement of submodels. Each submodel progresses its local time only if it is safe to do so, i.e. if it is certain that the submodel will never receive a message into its past. The submodel which cannot proceed is blocked. The historically first protocols were *asynchronous* in the sense that processes (submodels) proceed asynchronously of each other within the bounds dictated by the local causality constraint. There is no global information derived which would be used to control local simulation time advancement of submodels. These methods are thus more distributed in nature, as opposed to approaches with elements resembling more centralized view of the simulation.

The null message protocol

This is the first synchronization method proposed independently by Bryant (Bryant, 1977), and Chandy and Misra (Chandy & Misra, 1979). Therefore, it is also called the *CMB protocol*. Here the logical processes (the submodels) are connected through reliable directed links. Messages are sent in timestamp order along each link. The model topology is static. The structure of an LP is shown in Figure 7-5. There is an input queue for each input link (IQ_1, ... , IQ_n), where incoming messages are buffered. There is also a clock variable associated with each input link (IT_1, ... , IT_n). Its value is equal to the timestamp of the first message in the corresponding queue, or the timestamp of the last message transmitted along the link, if the queue is empty. The output messages wait in output queues (OQ_1, ... , OQ_m) to be sent. The time up to which output messages can be sent is maintained in the variable *OT*. The reason for messages waiting in output queues is to obey the rule of timestamp order of messages sent along a link. The LP has also the local event list *EL*, local state *S*, and maintains its local simulation time T_{sim}.

[12] In (Fujimoto, 2000), Fujimoto states that the local causality constraint is the *sufficient and necessary* condition of causal correctness. In fact, it is not necessary, as pointed out in (Ferscha, 1994), page 10: "... two events occurring within one and the same LP may be concurrent (independent of each other) and could thus be processed in any order."

The LP repeatedly selects the input link with the smallest clock value, and if its queue is non-empty, removes the message from the head of the queue. This message and local events with timestamps not later than the message are processed in order. If the queue with the smallest clock is empty, the LP must block until a message arrives which updates the clock value. Then a new smallest clock queue is selected, and the procedure repeats.

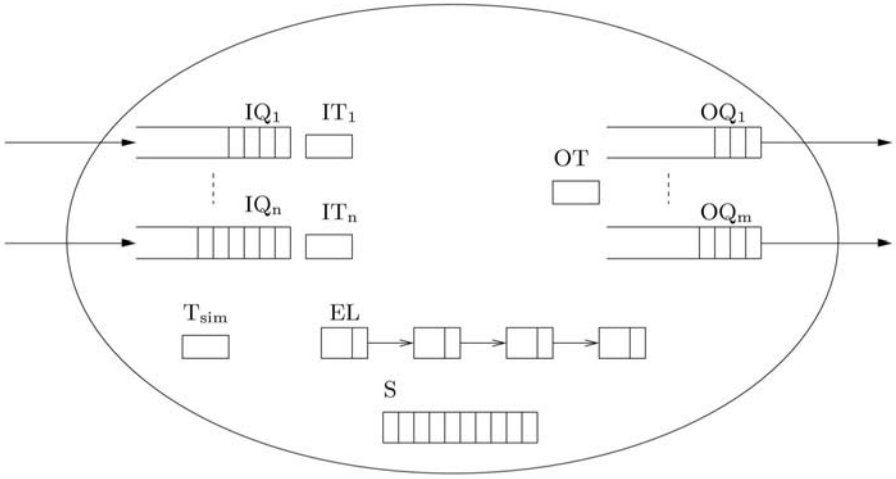


Figure 7-5. Logical process structure.

Waiting for messages and merging them with local events into a single time-ordered sequence ensures that the local causality constraint is obeyed. However, blocking of LPs on empty queues has the following consequence. If the model topology contains cycles (as many real models do) a circular blocking of processes can be formed. Processes involved in the cycle would block forever, thus the simulation (whole or a part) is in deadlock. To overcome the problem, the CMB protocol introduces *null messages*. These messages have no counterpart in the modeled system. They carry no information but a timestamp value which gives the receiving LP the lower bound of any future messages transmitted by the sender.

The timestamp of a null message is equal to local time plus some timestamp increment. The increment, called *lookahead*, represents the ability of the LP to look ahead into its future. This issue is more deeply studied in section 7.4. An important limitation of the null message protocol is that it cannot be used to simulate models having cycles with zero lookahead, i.e. when no LP in a cycle has positive timestamp increment. If a deadlock happens in such cycle, the null messages would circulate forever but none of the processes will advance its local time, thus the deadlock could not be broken.

In the original protocol, an LP sends null messages on each output link every time the local simulation time advances. This results in high volume of null messages which load the communication system and consequently degrade the performance of the simulation. It happens especially in model topologies with high fan-out. Several variants have been proposed to deal with this issue. In one of them, the LP delays sending a null message for some time interval. The null message is then sent only if no real message has been sent during that interval (Misra, 1986). In other variants, the null messages are sent only when the receiving LP demands them, when some of its input queues becomes empty (Su & Seitz, 1989). Although it takes longer time to obtain a

null message in the variants than in the original version, the variants can outperform the basic scheme because of the reduction in the number of null messages.

Another improvement of the basic scheme tackles the following problem related to loops in model topology. When a total timestamp increment of LPs along a loop is small, the null message circulates many times in the loop until its timestamp increases enough to allow to processes a real message. The *carrier null message* protocols in (Cai & Turner, 1990) expand the null message to include information of pending events in traversed LPs which is used to advance their local times more rapidly.

Deadlock detection and recovery

The setup in this scheme is the same as above, except the null messages. This method does not use null messages, nor any other mechanism for deadlock avoidance. The simulation (or its part) is allowed to deadlock, and a mechanism to detect and recover from this situation is provided (Chandy & Misra, 1981). The recovery mechanism is based on the observation that the event with smallest timestamp in the system can always be processed safely. The simulation runs as a sequence of parallel computations in which the parallel phases conducting the simulation and ending with deadlocks interleave with sequential "phase interfaces" resolving the deadlocks. Unlike the null message approach, this method can be used to simulate models containing cycles with zero timestamp increment (Fujimoto, 1990a).

This approach can be used for models where partial deadlocks can occur. It requires pre-processing of the model to identify the parts which can be engaged in a deadlock. Then the detection and recovery mechanisms are applied to those parts during the simulation itself (Misra, 1986, Fujimoto, 1990a).

The time of next event protocol

This protocol is intended for simulations where each processor hosts multiple LPs. The basic setup is like that of the deadlock recovery protocol. The time of next event (TNE) algorithm is designed to help unblock blocked LPs and resolve local deadlocks within a processor (Groselj & Tropper, 1988).

Consider an LP which is blocked because of some empty input queues. Let us assume that the smallest timestamp from the non-empty queues is smaller than the timestamp of any future message which will arrive to the empty input queues. Then the LP can process the smallest message present. The task of the TNE algorithm is to provide such lower bounds on future timestamps at the empty queues.

Assume a couple of adjacent logical processes LP_i and LP_j where the input queue at LP_j receiving messages form LP_i is empty. Within the TNE computation, LP_i determines the lower bound on timestamp T_i of the event that it will process next. If the actual event is not known at the moment, the local simulation time gives the safe lower bound. Then LP_i determines also the smallest timestamp increment $Tsmin_{ij}$ encountered by that event when sent to LP_j. Thus LP_j cannot expect a message from LP_i along that link with timestamp smaller than $T_{ij}=T_i+Tsmin_{ij}$. This information can be used by LP_j to update its T_j value, and the procedure can be applied recursively to the successors of LP_j which have empty input queues from LP_j. This leads to the shortest path computation on the graph of LPs connected by "empty links" finding estimates of the timestamps of next events on all empty links. If the estimate for a particular LP is higher than the LP's earliest event, the event can be safely processed, i.e. in effect the LP is unblocked.

The TNE algorithm runs when the processor is otherwise idle, i.e. when it cannot execute any simulation code itself. Running the algorithm can lead to unblocking some LPs so that execution of the simulation code can resume.

TNE doesn't address inter-processor deadlocks for which another algorithm is needed. An extension called *semi-global time of next event* (SGTNE) goes beyond the border of a particular processor, and helps to avoid inter-processor deadlocks too (Boukerche & Tropper, 1995).

7.3.2 Synchronous conservative methods

In the deadlock detection and recovery approach, the computation of a processor can be viewed as a loop consisting of two phases, in particular event processing and deadlock resolution. Several conservative synchronization methods have been designed with similar two-phase loop, but the transition between the phases is explicitly controlled here. The phase interface is realized using the mechanism of *barrier synchronization* known from general parallel programming.

A sketch of the algorithm executed by a processor is shown in Figure 7-6. In the first phase the processors identify which events are safe to process, and in the second phase these events are processed. After each phase the processors are synchronized. A processor executing the barrier primitive is blocked until all of the processor have executed the barrier primitive. This results in computation on all processors progressing in synchronous steps. The successive steps do not interfere with each other. This ensures that no events are being processed, and thus no new events are being scheduled when processors are busy with determining which events are safe to process. To achieve such separation of the phases, all messages must be received by their destination LPs in the same phase they have been sent. In other words, the barrier must block until there are no messages in transit, or no *transient messages*.

Methods dealing with transient messages are surveyed in (Fujimoto, 2000). Next we mention particular synchronous protocols, which differ in the way they determine safe events.

```
while( ! terminationCondition() )
{
     identify all events that are safe to process;
     barrier synchronization;
     process safe events;
     barrier synchronization;
}
```

Figure 7-6. Synchronous simulation algorithm.

The conditional event approach

In the *conditional event* approach (Chandy & Sherman, 1989a) the events on the event list as categorized are either *conditional* or *definite*. Definite events are going to occur in the simulation future without presumptions of those events that occur earlier in simulation time, i.e. definite events will not be canceled by earlier events. On the other hand, conditional events depend on the outcome of processing earlier events. Thus, in our previous terminology, the definite events are safe to process. Identifying safe events consists of two steps. In the first step, application specific knowledge is used by

each LP to determine its local definite events (e.g. highest priority job departure in a preemptive priority queuing server). Then in the second step, the earliest conditional event in the whole simulation is turned into a definite one, since there are no earlier events to affect it. To determine the earliest event, the global min-reduction operation is executed by the processors. After determining definite events the second phase follows in which they are processed.

The YAWNS protocol

A simple approach exploiting lookahead of LPs to determine safe events has been proposed within the simulator called YAWNS, standing for Yet Another Windowing Network Simulator (Nicol et al., 1989). Let the timestamp of earliest unprocessed event in LP_i at simulation time t be $T_i(t)$, and its lookahead be $LA_i(t)$. If the process receives no new messages, then the next message it will generate will have timestamp at least

$$\delta_i(t) = T_i(t) + LA_i(t) \tag{1}$$

Initially at simulation time t=0, we can predict that the earliest message will have timestamp

$$\delta(0) = \min_{\forall i}\{\delta_i(0)\} \tag{2}$$

All LPs can process events within the interval $\langle 0,\delta(0))$ independently. In general, at any simulation time t if all events prior to time t have been processed (in previous iteration), the time interval for safe events can be determined as $\langle t,\delta(t))$ where

$$d(t) = \min_{"i}\{d_i(t)\} \tag{3}$$

The time interval is also called *time window*. It moves along time axis as the simulation proceeds. In an iteration, the lower edge of the window is given by the upper edge from the previous iteration, and a new upper edge is determined by formula (3).

The only requirement to use this method is the availability of lookahead which is constituted by two actions in this case. First, an LP must send messages ahead of simulation time. Second, each logical process has to implement a function `Delta()`, which returns the timestamp $\delta_i(t)$ of next message sent by the LP provided it receives no new messages. This value is used in formula (3) for establishing the width of time windows. We will deal with the lookahead more in section 7.4. The synchronization method can handle simulation models with arbitrary topology which can even change during the course of the simulation. Messages need not be sent in timestamp order.

Unlike the conditional event approach, model specific knowledge is not directly used to identify safe events. Instead, the lookahead plays this role. It has been proven in (Nicol, 1993) that the construction of time window leads to safe event processing, i.e. any message generated during event processing has timestamp at least as large as the upper window edge. Such messages are beyond the current window because, by protocol definition, safe events have timestamps strictly less than the upper window edge $\delta(t)$. It implies, that any time window width must be strictly positive, i.e. for a window computed at simulation time t, every submodel must supply a value $\delta_i(t) > t$. Otherwise the protocol identifies no safe events and simulation cannot proceed anymore.

In the same paper (Nicol, 1993), the protocol has been analyzed and interesting results have been derived: the asymptotic (in terms of model size) time complexity of the average total overhead (including synchronization, lookahead calculations, processor idle time, and event list manipulation) per event is that of optimized serial simulator. In (Nicol, 1998), it has been shown that under some conditions the protocol is scalable in the sense that the overhead remains within constant factor if the model and the parallel architecture grow simultaneously.

The promising results of analytical studies of this protocol contributed to its use within really large scale simulations. To mention the most significant cases, YAWNS synchronization is used in the DaSSF simulator (Liu & Nicol, 2001) for network simulations of scale approaching the global Internet (Cowie et al., 1999), or in the SWAN simulator to simulate large scale mobile ad hoc networks (Liu, 2003).

Distance between logical processes

The simple synchronous protocol described above does not exploit model topology. It determines the time window width as global minimum of timestamps of all future messages. Consider a model where one of the LPs has significantly smaller lookahead than the other LPs. Then the window will be much shorter than necessary for most LPs. This results in many "empty" iterations, i.e. iterations where the LPs will have no safe events to process. In this case the synchronization method does not exploit the available lookahead which leads to performance degradation. In general, this happens in models where the lookahead of individual LPs differs significantly.

The method based on distance between LPs addresses this issue (Ayani & Rajaei, 1992). The distance D_{ij} between LP_i and LP_j is the lower bound on the amount of simulation time which elapses until a state change in LP_i propagates and affects the state of LP_j. It is determined as the shortest path from LP_i to LP_j in the model graph where the vertices represent logical processes and edges represent event message flow and are weighted by the corresponding minimum scheduling delays. The distances can be organized in a square matrix called the *distance matrix* where the element in row i and column j represents the distance D_{ij}.

The matrix is used to determine the lower bound on the timestamp (*LBTS*) of any future message a logical process can receive. Let T_i be the timestamp of the earliest pending event in LP_i, or ∞ if there are no events in LP_i. Then

$$LBTS_i = min_{\forall j}\{T_j + D_{ji}\} \tag{4}$$

Events within LP_i with timestamp less than $LBTS_i$ are safe. It should be noted that the computation includes the case of $j=i$. The most serious drawback of this method is that each LP needs to communicate to all other LPs in order to obtain their T_i value. For N logical processes, this generates N^2 messages to determine safe events in each iteration. Such overhead prevents the method from scaling to large size models.

Bounded lag

In order to reduce the large communication overhead incurred in the previous protocol it is useful to distinguish near future from the far future. Because of the cause-effect principle, events in the far future are more likely to be unsafe than those in the near future. Thus the effort to determine safeness of far future events is likely to be wasteful. Let T_S be the timestamp of the earliest pending event in the entire simulation, and BL be a constant. Then T_S+BL is the border separating near future from far future. Far

future events are never considered for processing in the present iteration. This ensures that local simulation times of LPs are apart from each other not more than *BL* which is the *bounded lag* (Lubachevsky, 1989). As a consequence, when LP_i determines the lower bound on timestamp of messages it will receive in the future, it need not consider such processes LP_j for which $D_{ji} > BL$ since such input messages fall behind the border T_S+BL. Then the new formula for $LBTS_i$ is

$$LBTS_i = min_{\forall j: D_{ji} \leq BL} \{T_j + D_{ji}\} \quad (5)$$

which restricts the set of LPs from which T_j needs to be collected.

It should be noted that while in the previous protocol the $LBTS_i$ value computed according to (4) serves as the upper bound of safe timestamps processed by LP_i in the current iteration, it is not the case here. Using (5), $LBTS_i$ can be larger than T_S+BL. Thus in order to maintain the bounded lag, the upper bound of safe timestamps is given as $min\{LBTS_i, T_S+BL\}$. This requires the global minimum reduction to compute T_S and broadcasting it to all LPs. Such operations can scale quite well, unlike the all-pairwise communication in the previous protocol.

The quantity of bounded lag is sometimes referred to as *time window* (Fujimoto, 2000). Here the meaning of this term is different than in the context of YAWNS mentioned above. An important question concerns setting the size of the lag. Larger values lead to more asynchrony and thus more events in each iteration, but require larger overhead to determine safeness of the events. On the other hand, smaller values allow smaller number of concurrently processed events. In general, the simulation needs to be tuned for each application to set the lag size yielding the best performance.

Conditional versus unconditional information

In this paragraph we would like to point out the effect of transmitting unconditional versus conditional information among LPs in the conservative protocols. An example of unconditional information is the lower bound on timestamp carried by the null messages in CMB protocol. This is a "promise" of the sending LP that will be kept unconditionally. As an opposite example, consider the YAWNS protocol where the δ_i value represents the timestamp of the earliest future output message sent by LP_i *provided that LP will receive no new input messages*. Thus δ_i is conditioned on the fact that LP_i could receive a new input message with timestamp smaller than its current earliest local event. Without the conditional information, as in the CMB scheme, LPs only advance in lookahead increments and multiple iterations are needed to advance the time to the timestamp of the next unprocessed event. Using the conditional information allows to avoid the problem and to jump directly to the next unprocessed event.

7.3.3 Optimistic methods

In contrast to conservative protocols, the optimistic methods take the complementary approach to causality violations. They admit causality errors temporarily, and provide support for detecting and correcting them. The main advantage is that parallelism can be exploited in situations where causality errors are possible, but in fact do not occur. In the pioneering work of this area (Jefferson, 1985), the notion of *virtual time* was introduced as an analogy to virtual memory systems known from operating systems. A page in virtual memory is analogous to an event in virtual time. The virtual address

of a page is its spatial coordinate, the virtual time of an event is its temporal coordinate. All pages in the virtual memory can be accessed, however access to those pages resident in the main memory is much less expensive than those swapped out and causing a page fault. Similarly in the time domain: events at any virtual time can be processed. However processing events in the virtual future is relatively inexpensive, while events in the virtual past cause very expensive *time fault*, i.e. *rollback*. Numerous other illustrations of the analogy are mentioned in (Jefferson, 1985).

The paper introduces also an implementation of the principle called *time warp*. Before elaborating its main features, let us mention another important contribution which shed light into the foundations of PDES synchronization.

Spectrum of options for parallel simulation

Categorization of PDES synchronization methods to conservative or optimistic gives just a simplified characterization. In (Reynolds, 1988), Reynolds introduces a much finer classification of synchronization methods. He defines a system of orthogonal *design variables* constituting a *design space*. Particular synchronization methods represent points in this space. There are nine design variables in total. Two of them are most often cited in later PDES literature. These design variables are as follows:

- Aggressiveness – processing events regardless of whether they are safe to process or not. Synchronization methods can employ various degree of aggressiveness. In the extreme case, if a message arrives at an LP which has no other events to process, it processes the message immediately.
- Risk – transmitting messages resulting from aggressive or conditional processing in an LP. A message containing an event which is not safe is a risk message.

While aggressiveness is related to intra-LP processing, risk deals with inter-LP communication. Example of risk messages includes the messages containing conditional information in conservative methods (see section 7.3.2). However, risk is typically mentioned in conjunction with aggressiveness within the time warp mechanism, elaborated below. Some protocols support aggressive event processing within individual LPs, but do not propagate possibly erroneous messages, i.e. they lack risk. An example is the Breathing Time Buckets protocol mentioned later.

Time warp

Time warp is the most widely used optimistic approach originating from the work of Jefferson in (Jefferson, 1985). Pure time warp combines unbounded aggressive event processing with risk messages. It supports access to past events, as required by the notion of virtual time. This is necessary to correct causality violations which may result from the optimistic processing.

The basic setting is similar to the conservative methods. The model consists of logical processes which communicate exclusively by exchanging timestamped messages. Communication lines are assumed to be reliable. Processes may send messages in arbitrary order with respect to their timestamps, and message delivery need not be order-preserving. Processes may be created and destroyed dynamically. To control the simulation, the simulation executive contains two parts, namely the *local control mechanism*, and *global control mechanism*. The local control is encapsulated within each process, while the global control mechanism requires distributed computation involving all LPs.

The skeleton of local control within an LP is the event processing loop, similarly to sequential simulation. Unlike conservative mechanisms, safeness of events is not determined. Events are aggressively (optimistically) processed and local simulation time is advanced. This may result in receiving a message with timestamp less than the local time, called *straggler message*. This indicates causality violation within the LP. To correct it, all events with timestamp larger than the straggler must be "undone", and that part of state trajectory must be recomputed including the straggler. Undoing incorrectly processed events is called *rollback*, and it has to eliminate all effects of the events which may be twofold:

- changes of local state variables
- scheduling new events and sending messages to other LPs

To rollback state variables, periodical state saving is necessary, referred to as *checkpointing*. There are two variants of checkpointing. In *copy state saving*, a snapshot of the entire local state is taken and saved. On the other hand, *incremental state saving* maintains a record of changes to individual state variables. The former method is more memory demanding, but it is easier to implement, and state restoration is more effective. For the latter method the opposite is typical. Recently, a new interesting technique for state restoration has been invented as an alternative to checkpointing, called *reverse computation* (Carothers et al., 1999).

Undoing the effect of erroneously sent messages is accomplished by means of *anti-messages*. Upon receiving an anti-message, there are three cases possible for the receiving LP:

- The corresponding positive message has not been processed yet. The two messages annihilate each other.
- The corresponding positive message has already been processed. The anti-message is a straggler causing a secondary rollback.
- The positive message has not been delivered yet. The receiving LP temporarily ignores the anti-message. When the positive message arrives, the two are annihilated.

There are two variants of how an LP can cancel messages it has previously sent. With *aggressive cancellation*, upon receiving a straggler message, the LP immediately sends anti-messages for all positive messages which resulted from events being rolled back. On the other hand, the *lazy cancellation* strategy first processes the straggler and re-processes the rolled back events. It then compares the messages generated before the rollback with those resulting from event re-processing, and cancels just those that do not match.

The local control mechanism just outlined is sufficient to ensure causally correct event processing. However, there are two problems which it does not deal with:

- The computation consumes memory, especially for state saving which is never released. A mechanism is required which would reclaim memory used for information that is no longer needed. Such memory reclamation is called *fossil collection*.
- Simulation code may include operations such as I/O operations that cannot be rolled back. The local control mechanism does not tackle this issue.

These two points provide motivation for the global control mechanism. Both points can be solved by determining the lower bound on the timestamp of any future rollback. Such lower bound is referred to as *global virtual time* (GVT). Having this value, memory holding information with timestamp less than GVT can be reclaimed, because no rollback in any LP will go to virtual time less than GVT. Similarly, since I/O computation with virtual time lower than GVT will never be rolled back, it can be safely committed. GVT at real (wallclock) time T during a simulation run can be defined as the minimum timestamp of all unprocessed and partially processed messages and anti-messages in the entire simulation at real time T. GVT is periodically computed in the course of the simulation using one of the known distributed GVT algorithms. The algorithms are surveyed in (Fujimoto, 2000). There are numerous variants, modifications, and improvements to the original time warp mechanism. The objectives of them are better memory management and better performance achieved through various techniques. Elaborating them is beyond the scope of this document. A good survey is given in the book (Fujimoto, 2000).

Breathing Time Buckets

The Breathing Time Buckets (BTB) is an optimistic, risk free synchronous protocol used within the SPEEDES simulator, standing for Synchronous Parallel Environment for Emulation and Discrete Event Simulation (Steinman, 1991).

The computation proceeds in iterations, where each iteration consists of several steps executed synchronously over all of the processors. Assume that all events with timestamp less than t have been processed and processors are globally synchronized at global virtual time t. The steps of an iteration at processor i are as follows:

1. Optimistic processing of local events in timestamp order. Event messages for other processors generated as a result of this processing are not sent in this step. Rather they are kept in an auxiliary buffer. The minimum timestamp $h_i(t)$ of these messages called *local event horizon* is computed after generating each new message. This step is done when all events with timestamps prior to the local event horizon have been processed. If no messages are generated, $h_i(t)=\infty$.
2. Computing of the (global) *event horizon*

$$h(t) = \min_{\forall i} \{h_i\{t)\}$$

   and broadcasting it to all processors.
3. Sending messages which have been generated by events with timestamps less then the event horizon $h(t)$. Since $h(t)$ is the timestamp of the earliest message sent in the current iteration, events prior to $h(t)$ can be safely committed, and messages generated as a results of processing these events can be transmitted. It can be guaranteed that these messages will not be rolled back.
4. Resolving processed events beyond the event horizon. The minimum timestamp of messages received in the previous step is determined. Events with timestamp larger than this value have to be rolled back, and the corresponding generated messages discarded from the auxiliary buffer. Other processed events beyond the event horizon are kept, they can save some computation in the next iteration, if confirmed correct.
5. Advancing the global virtual time t to the event horizon $h(t)$.

There is an obvious similarity between BTB and the YAWNS protocol mentioned earlier. Both are synchronous and determine a global time window in each iteration. The principal difference is that YAWNS requires explicit lookahead of LPs to establish the time window, while BTB determines it via optimistic event processing, thus explicit lookahead is not needed here.

7.3.4 Comparison between conservatism and optimism

There have been many attempts to compare the conservative and optimistic methods in the history of PDES. Yet there is no simple, definitive answer which approach is better. Moreover, the overall experience indicates that no such answer is to come any soon. This is because the performance of a parallel simulator largely depends on the characteristics of the model, and there is no single linear performance metric that could be universally used. Thus, rather than giving any such answer, this section briefly summarizes pros and cons of both classes of approaches (Liu, 2003).

From the software engineering point of view, design and implementation of a conservative simulator is simpler in comparison to an optimistic one. The difficulties with optimistic execution are not just due to state saving and rollback themselves. The pure Time Warp usually does not provide acceptable performance. It requires a number of additional sophisticated techniques to keep its optimism and memory consumption within reasonable bounds so that performance does not suffer. These improvements increase the complexity of the implementation. The GVT computation must be executed “on the fly”, i.e. without freezing the entire simulation which is not easy to run well. Another source of difficulties stems from the inherently hard debugging and fine-tuning of an optimistic simulator. On the other hand, conservative simulators requiring deadlock detection and recovery, or barrier synchronization and global reduction operations are relatively easy to implement and fine-tune.

The effort needed for model creation is another point worth to notice. In conservative simulators, the lookahead must be extracted and explicitly specified by the modeler to achieve reasonable parallel performance. On the other hand, optimistic methods exploit the available parallelism automatically by the essence of the algorithm. However, this is at the expense of the overhead including state saving and rollbacks. In other words one can say, that in optimistic methods, the cost of extracting the lookahead is included in the runtime overhead, while in conservative approaches this cost is shifted to the phase of model creation and is borne by the modeler. However, the latter case requires that the model allows to extract the lookahead prior to (i.e. without) actually simulating it.

In simulation of large scale models, memory usage becomes an important point. Since optimistic processing needs more memory, it can more easily consume the available physical memory of a computer. In other words, the maximum size of models in an optimistic simulator has lower limit than in a conservative simulator. There are storage optimality algorithms for the optimistic methods which can cut down their memory usage, however at the expense of performance. As a result, if there is sufficient lookahead, conservative methods are more suitable for very large scale models. Important applications falling into this category include computer architecture and communication networks simulations.

Traditionally, the issue attracting major attention in the context of PDES is the performance in terms of time or speedup. As will be elaborated later in section 7.4,

conservative methods require good lookahead to achieve good performance. The amount of lookahead determines the extent to which conservative methods can exploit model parallelism. Even potential dependencies between events assigned to different processors limit the degree of independent event processing the processors may afford.

On the other hand, model parallelism can be fully exploited in optimistic simulation in that the parallelism is limited only by actual dependencies instead of potential ones. In other words, the optimistic methods profit especially in situations when causality violations are possible but the probability that they really happen is low. In such cases, conservative methods must block, but the aggressive optimistic processing succeeds thanks to the low probability of rollback. If, however, the probability increases, the performance penalty due to more rollbacks can diminish, or even outweigh the larger extracted parallelism.

As can be seen from this discussion, the choice of the more suitable approach depends largely on the characteristics of the application. This led to the idea of constructing simulators providing multiple synchronization methods, so that the same model may be simulated using either of the available synchronization algorithms, e.g. (Perumalla, 2005).

7.4 The lookahead

As our interest is aimed at lookahead related issues, we devote this separate section to the introduction of the important knowledge relevant to this topic. To reason about the necessity of prediction or lookahead for conservative synchronization, consider a simulation with no lookahead. In such situation, an LP or submodel with local time t can send a message with timestamp just t. It follows that the submodel with globally smallest local time t_{min} can generate messages with timestamp t_{min}, preventing other submodels from processing events with timestamps larger than t_{min}. This would result in serial execution of events.

This consideration indicates the importance of lookahead. Many studies, both analytical and empirical, have shown the importance of lookahead for the performance of conservatively synchronized simulations, e.g. (Preiss & Loucks, 1990, Fujimoto, 1990a, Wang & Abrams, 1995, Meyer & Bargrodia, 1999). This is in accordance with intuition: the larger the lookahead, the longer the simulation time interval in which submodels may run independently, and/or the less the probability of submodel blocking.

7.4.1 Lookahead ability

Roughly speaking, lookahead is the ability of a LP or submodel to predict its simulation future. More precisely, as mentioned in the the discussion on conservative synchronization protocols, a logical process at local simulation time t having lookahead L will not sent any message (real or null) with timestamp less than $t+L$. In general, the lookahead may vary in the course of simulation, so it is more appropriate to express it as $L(t)$, and the earliest timestamp as $t+L(t)$.

Generally, this ability is achieved through sending messages ahead of simulation time. It can be nicely illustrated using the notation of event graphs. Consider a submodel modeling a non-preemptive G/G/1 queue with arbitrary queuing discipline depicted in Figure 7-7.

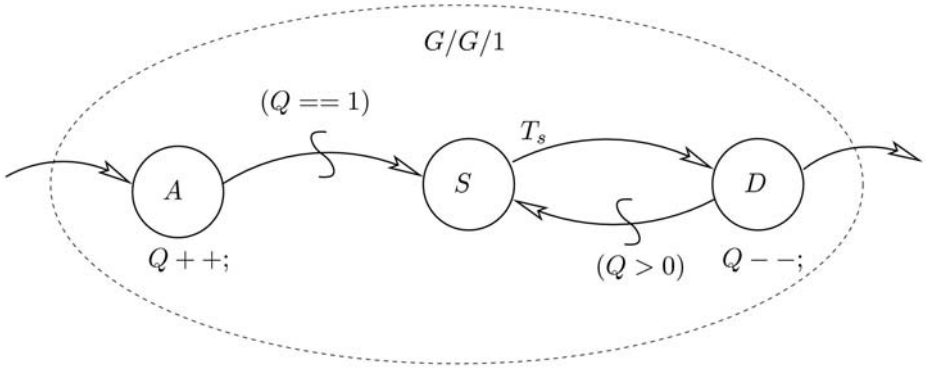


Figure 7-7. Zero lookahead G/G/1 queue.

The submodel consists of three events: job arrival (*A*), start of service (*S*), and job departure (*D*). The outgoing arrow represents scheduling an event due to the job departure from the queue. The scheduled event is not shown in the picture because it belongs to another submodel. In PDES terms, the arrow represents an output message sent by this submodel. In sequential simulation, the submodel is naturally constructed so that output message is sent by the departure event. It means that the difference between the timestamp of the message, and simulation time at which it is sent, is zero. Therefore such submodels are referred to as *zero-lookahead* submodels. The lookahead of this model as a function of simulation time is shown in Figure 7-8. The piecewise constant graph depicts the $t+L(t)$ function, while the piecewise linear graph with slope -1 shows the lookahead $L(t)$ itself. Its shape follows from the fact that $L(t)$ decreases with the same rate as the simulation time progresses towards the point when the next output message is sent. If a particular synchronization method needs to specify a single lookahead value, the minimum of $L(t)$ has to be taken which is zero in this case. This is the consequence of the zero-lookahead property of the submodel.

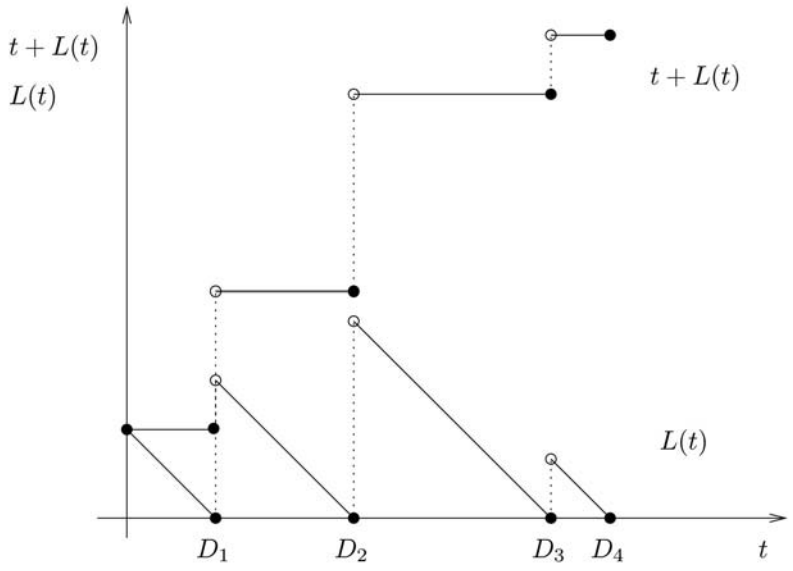


Figure 7-8. Zero lookahead G/G/1 queue - functions t+L(t) and L(t).

Message pre-sending means sending the output message from an earlier event. A version of the submodel with better lookahead sends the output message from the *S* (start-of-service) event, see Figure 7-9.

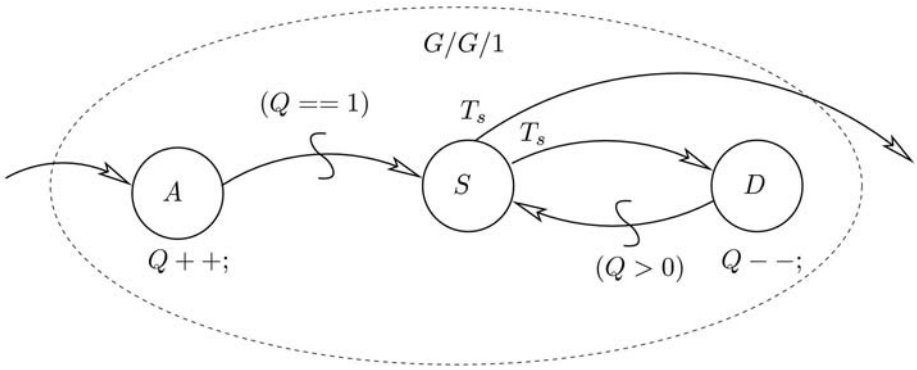


Figure 7-9. G/G/1 queue with message pre-sending.

Thus, when a job enters service, an output message with timestamp equal to its departure time is sent which "reports" that job's departure. This is possible because of the non-preemptive property of the queue. Another assumption of message pre-sending is that the destination of a job leaving this queue can be determined ahead of time (at the instant of service start). Then, as a consequence, immediately after the *S* event, the $L(t)$ value is determined by the departure time of the next job whose service has not yet been started. Figure 7-10 shows the lookahead of the submodel. The minimum value of $L(t)$ is equal to the minimum service time in the queue.

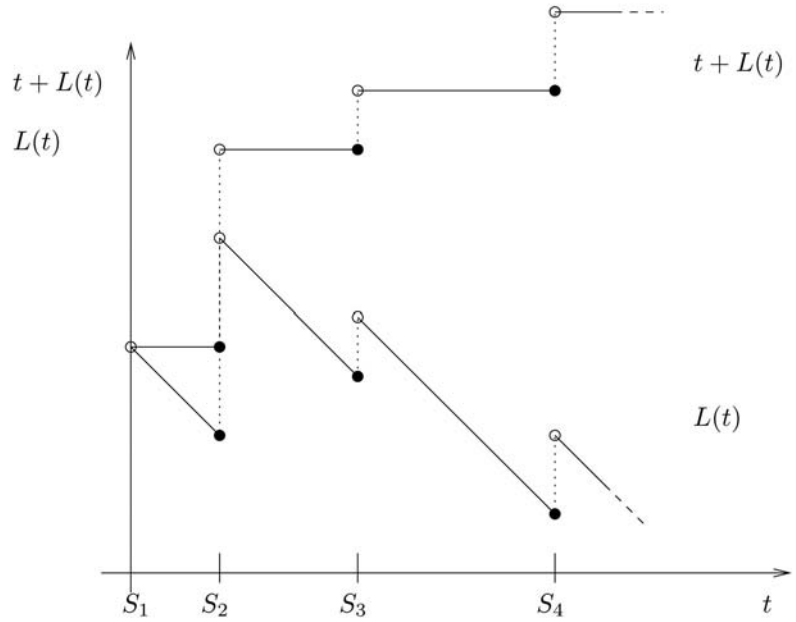


Figure 7-10. G/G/1 queue with message pre-sending – functions t+L(t) and L(t).

In the above example, the messages sent out from the submodel are real ones, i.e. they carry actual events. In the CMB protocol, the null messages are also sent ahead of time. Recall that a null message sent at simulation time t has timestamp $t+L(t)$ where $L(t)$ is the lookahead, c.f. CMB protocol in section 7.3.1. In the example submodel here, the single lookahead value required by CMB protocol is equal to the minimum service time in the queue *minServ*. When processing a safe *A* or *D* event, a null message with timestamp *t+minServ* can be sent out, provided it is larger than the timestamp of the last output message. The *S* event need not send null messages, since it sends real ones.

The source of lookahead depends on the particular model. Some examples of where lookahead may come from are as follows:

- Limitations in the modeled system how quickly can physical processes interact with each other. In air traffic simulation this is the time needed for an aircraft to fly from one airport to another. In communication network simulation this is the time a message needs to be delivered from one node to another. In logic simulation this is the time necessary for a signal to propagate from one gate to another.
- Limitations how quickly an LP can react to a new event. In air traffic simulation this is the time an aircraft must remain on the the ground to exchange passengers before it takes off again. In communication network simulation this is the time a message is delayed in a node. In logic simulation this is the time delay needed for a gate output to react to a value change on that gate's input.
- Non-preemptive behavior. Consider that in the air traffic example, once an aircraft departed from one airport, nothing can prevent it from arriving at its destination airport at the assigned arrival time. Such behavior can be used to derive lookahead. Typically, non-preemptive service of servers is the source of lookahead in queuing system simulations.

The task of deriving the lookahead of a particular model for a particular synchronization scheme is hard to automate, and typically is solved by the simulation designer.

7.4.2 Lookahead ratio

The ability of a submodel to predict the future can be quantified by the $L(t)$ function, or simply by its minimum value. This is an absolute quantification. However, the lookahead ability is better expressed by the *lookahead ratio*, which also better correlates to the simulation speedup (Preiss & Loucks, 1990). The notion of lookahead ratio (*LAR*) evolved over time and several, slightly different *LAR* definitions can be found in PDES literature, e.g. (Fujimoto, 1990b, Wagner & Lazowska, 1989, Preiss & Loucks, 1990). We use the latest one of them from (Preiss & Loucks, 1990). The definition is based on three characteristic (simulation) times for a message passing through a submodel:

- t_{cause} At this time the decision to send an output message is first made. Between t_{cause} and the actual sending of the message, its departure time (or even its departure itself) may be altered. If an outgoing message is caused by more than one incoming message, t_{cause} is viewed as the last of these cause messages.

- t_{commit} This is the earliest time after which no arriving message can prevent or alter the departure of the message in question. Thus, t_{commit} is the time when the submodel can send the outgoing message.
- t_{effect} This is the timestamp of the outgoing message.

Figure 7-11 shows these three times for a typical message.

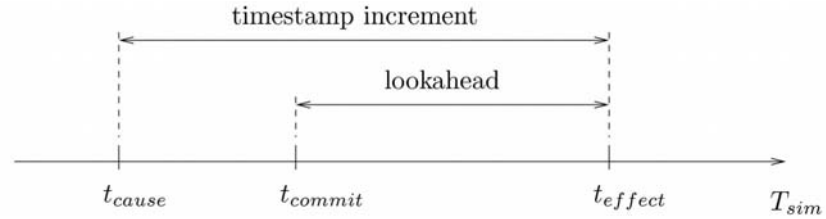


Figure 7-11. Characteristic times for LAR definition.

Using the times, lookahead ratio is defined as follows

$$LAR = \frac{E(t_{effect} - t_{cause})}{E(t_{effect} - t_{commit})} \tag{6}$$

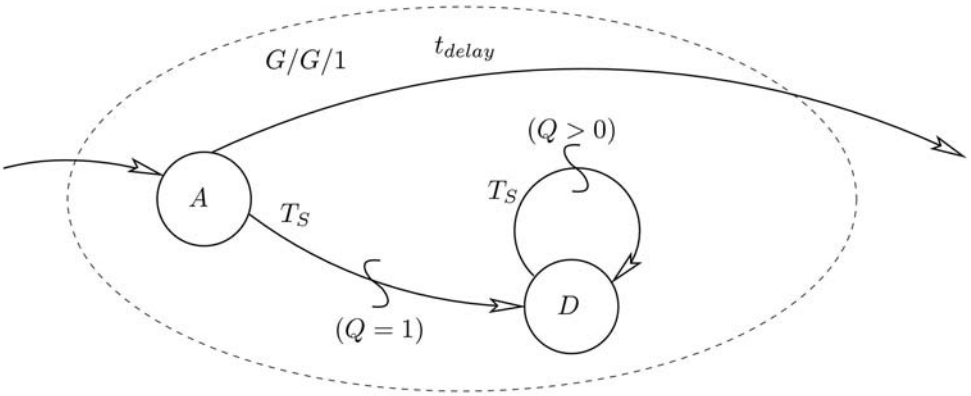
Since $t_{cause} \leq t_{commit} \leq t_{effect}$, *LAR* lies in the interval $\langle 1,\infty)$. According to formula (6), the larger *LAR*, the smaller the lookahead ability. Because of this inverse proportional property, and also to avoid the problem with infinity, the reciprocal of *LAR* is sometimes used, called *inverse lookahead ratio* (*ILAR*). It is defined as *ILAR*=1/*LAR*. The quantity of *ILAR* is in the interval $\langle 0,1\rangle$, and the larger value corresponds to better lookahead ability. This makes *ILAR* a more intuitive metrics, and this is why we prefer to use it in the following text.

To illustrate these notions, let us consider a submodel simulating a M/M/1 queue with FIFO queuing discipline which sends the output message at the instant of service start. The behavior of the submodel can be described by the event graph shown in Figure 7-9. In this submodel, the decision to send an output message is first made when an input message arrives, i.e. $t_{cause}=ts(A)$. The output message is committed (sent) when the event *S* occurs, thus $t_{commit}=ts(S)$. Finally, the timestamp of the output message is $t_{effect}=ts(D)$. To derive the inverse lookahead ratio of this submodel we exploit the markovian property of customer arrivals and service time distribution for which analytical solutions are known. Let the customers arrive with intensity λ, and the server provides service with intensity μ. Let the queue be stable, so that $\lambda<\mu$. We can write

$$ILAR_{M/M/1} = \frac{E(ts(D) - ts(S))}{E(ts(D) - ts(A))} = \frac{\frac{1}{\mu}}{\frac{1}{\mu - \lambda}} = \frac{\mu - \lambda}{\mu}$$

The numerator is the mean service time $1/\mu$, and the denominator is the mean response time $1/(\mu-\lambda)$ of the queue. As can be seen from the above formula, smaller λ leads to larger *ILAR*. This is because with smaller arrival intensity the queue gets less loaded, which leads to shorter mean response time (the timestamp increment in Figure 7-11) while retaining the same mean service time.

Let us further elaborate the M/M/1 submodel. The fact that the server is non-preemptive together with the FIFO queuing discipline can be used to further improve the lookahead. If the destination of a job departing from this queue can be determined at the time the job arrives, the output message can be sent as early as the cause input message arrives. Thus $t_{commit}=t_{cause}=ts(A)$ and the denominator equals the numerator, resulting in *LAR*=*ILAR*=1. This is the largest lookahead ability a submodel can achieve. We will refer to such submodels as LAR1 (ILAR1) submodels. The behavior of the LAR1 M/M/1 submodel can be described by the event graph depicted in Figure 7-12. In fact, the LAR1 property is possible for any non-preemptive G/G/1 queue with FIFO discipline. To simplify the event graph, we eliminated the *S* event. The code of events is given in the attached table. The variable t_{LD} stores the timestamp of the last output message sent. *SERV* is a queue keeping the service times of jobs for which the messages reporting their departures have been pre-sent, but the jobs have actually not been processed yet.



event	code
A	$Q++$; $T_S = RND.Next()$; if $(Q > 1)$ $SERV[tail] = T_S$; if $(t_{LD} > ts(A))$ $t_{LD} = t_{LD} + T_S$; else $t_{LD} = ts(A) + T_S$; $t_{delay} = t_{LD} - ts(A)$;
D	$Q--$; if $(Q > 0)$ $T_S = SERV[head]$;

Figure 7-12. G/G/1 queue with LAR=1.

7.4.3 Dimensions of lookahead

Different simulation models may offer different forms of lookahead. On the other hand, individual synchronization protocols require lookahead specification in specific ways. In order to provide a "roadmap" in these subtleties of lookahead, Nicol introduces a classification of lookahead *dimensions* (Nicol, 1996). It is a system of orthogonal properties of predictions made in the model:

- Time / Content Lookahead. A submodel SM_i may be able to determine that it will not affect another submodel SM_j at a time less than t. If SM_i has a better

lookahead ability, it may be able to determine also how it will affect SM_j. In the former case, the prediction contains only the temporal aspect (e.g. in the form of a null message), resulting in time lookahead. In the latter case, the prediction provides also some content (real message) giving SM_j the opportunity to use the prediction in a more advanced way.

- Bounded / Exact Time Lookahead. A submodel SM_i has bounded-time lookahead if it is able to predict that it will not affect SM_j at time *less* than t. It is different from the situation when SM_i can predict that it will not affect SM_j at time less than t and it *will* affect SM_j *at* time t. In the latter case, SM_i has exact-time lookahead. If SM_i allows to combine exact-time lookahead with content lookahead, it can actually deliver the predicted event to SM_j.
- Directed / Semi-directed / Undirected Lookahead. If a submodel SM_i is able to predict that the destination of its future interaction is a particular submodel SM_j, then SM_i has directed lookahead. If the forecast interaction may affect a subset of submodels, the lookahead is semi-directed. Finally, if SM_i knows only that it will affect any submodel, its lookahead is an undirected one.
- Conditional / Unconditional Lookahead. This dimension of lookahead is related to the conditional information mentioned in section 7.3.2. The idea of conditional lookahead is that submodel SM_i knows that it will not affect SM_j before time t, provided the state of SM_i does not change before time t. The conditional lookahead allows to advance simulation time more rapidly in comparison to unconditional lookahead.

The cartesian product of the dimensions forms a space of options for lookahead. Synchronization protocols have different needs of lookahead dimensions, thus they occupy different points in the space. Some protocols may be able to support multiple points in this space.

7.4.4 Lookahead in the YAWNS protocol

As already described in section 7.3.2, the lookahead is constituted by two complementary actions. The first one is message pre-sending. The second one is for a submodel SM_i the ability to predict the timestamp $\delta_i(t)$ of the next output message provided the submodel receives no further input messages. Thus $\delta_i(t)$ is a *conditional* timestamp, and YAWNS uses conditional lookahead. If $\delta_i(t)$ is the exact time of next message, the lookahead is exact-time. The protocol works also if $\delta_i(t)$ is a lower bound on the timestamp, in which case the lookahead is bounded-time. The protocol does not distinguish the potential destination of a message, thus any submodel can be affected by a message. This results in undirected lookahead.

To illustrate the YAWNS lookahead, let us consider a submodel modeling a non-preemptive G/G/1 server. We show the submodel in two different settings. First with smaller lookahead, i.e. a G/G/1/ submodel sending output messages at times of service start, as depicted in Figure 7-9. Later we will elaborate the same queue having LAR=1, as shown in Figure 7-12. In both cases, we are interested in computing the time $\delta_i(t)$ for the particular pre-sending method.

In the former case, to calculate $\delta_i(t)$, we need the information about arrival events scheduled for the queue in the local event list, and the state of the server (busy or idle) (Nicol, 1993, Solčány, Šafařík, 2001). Further, it is necessary to pre-sample service times.

The following rules hold:

- If the event list contains no arrival events scheduled for the queue, $\delta_i(t)=\infty$. This follows from the assumption that no new messages will arrive.
- If there are some arrival events scheduled and the server is idle, $\delta_i(t)$ is the time of the departure of the next job to be given service, assuming that no further arrival events will be inserted into the event list. In particular, one of the arrival events is selected according to the queuing discipline, and $\delta_i(t)$ is the sum of the timestamp of the selected arrival event plus the corresponding pre-sampled service time.
- If there are some arrival events scheduled and the server is busy servicing a job, $\delta_i(t)$ is the departure time of the next job to receive service after the current one completes, assuming that no further arrival events will be scheduled. Let t_{LD} be the departure time of the job receiving service. Two cases are possible:
  - If there are some arrivals scheduled for time less than t_{LD} then $\delta_i(t)=t_{LD}+T_S^*$, where T_S^* is the pre-sampled service time of the job selected according to queuing discipline from among the jobs with arrival time less than t_{LD}.
  - If all the arrivals are scheduled for time greater than t_{LD}, then let A^* be the earliest of these arrivals, and let T_S^* be the corresponding pre-sampled service time. The time of next message $\delta_i(t)=ts(A^*)+T_S^*$.

If there is a lower bound of the service time known for the queue, it can be used in the computation of $\delta_i(t)$ every time instead of the actual pre-sampled service time. However, it would lead to smaller lookahead. Now let us deal with the case when the queuing server submodel has *LAR*=1 (Solčány & Šafařík, 2003). Here, beside the non-preemptivness also the FCFS (First Come First Served) order of customer servicing is necessary. The event graph of the submodel is depicted in Figure 7-12. To compute the value of $\delta_i(t)$, we again need to know the arrival events scheduled for the queue, and we need to pre-sample service times. The rules for computing $\delta_i(t)$ are as follows:

- If the event list contains no arrival events scheduled for the queue, $\delta_i(t)=\infty$. This follows from the assumption that no new messages will arrive.
- If there are some arrival events scheduled, the next output message will be sent as a result of processing the earliest of the arrivals. Let A^* be the earliest of the arrivals, and let T_S^* be the corresponding pre-sampled service time. Let t_{LD} be the timestamp of the last output message sent by the submodel. Then $\delta_i(t)=max(ts(A^*), t_{LD})+T_S^*$.

Message pre-sending in a submodel is the prerequisite of successful prediction of reasonable future conditional timestamps, and thus a necessary condition for the YAWNS synchronization. As a counter-example, consider a submodel with no pre-sending, as in Figure 7-7. The first occurrence of the *D* event sends a message with timestamp *ts*(*D*) which is the $\delta_i(t)$ value of the submodel and determines the upper edge of a time window. However the *D* event is not processed within the time window (the time window interval is right-open). Thus the message is not sent within the window, and it must be taken into account when establishing the next window. This happens to be at time *ts*(*D*), and the timestamp of the message is also *ts*(*D*). The result is a zero-length time window which is prohibited by the protocol (c.f. the explanation of YAWNS synchronization in section 7.3.2). Therefore every submodel must pre-send messages, and must have non-zero lookahead.

7.5 Discrepancies between DES and PDES

After almost a decade and a half of quite successful developments in the area of PDES, the PDES community observed that in spite of the successes, it has had little impact in the general discrete event simulation community. This triggered an effort of self assessment and a discussion about reasons of such situation. The first and richest part of the discussion took place in a series of articles within the summer 1993 issue of the *ORSA Journal on Computing (JoC)* (Bagrodia, 1993, Abrams, 1993, Unger & Cleary, 1993, Fujimoto, 1993b, Fujimoto, 1993a, Reynolds, 1993, Lin, 1993). It then continued in the keynote talks at PADS Workshop in 1996 by R. Fujimoto (Fujimoto, 1996), and in 1997 by D. Nicol (Nicol, 1997a). All these contributions are summarized in subsection 7.5.1. Another view is given by the leading personalities in modeling methodology in (Page & Nance, 1994).

The main points of this article are included in subsection 7.5.2. There is one more paper dealing with this issue by E. Page (Page, 1999). It looks at previous PDES assessments and considers some contemporary directions in distributed simulation like military simulation systems and web-based simulation and their possible relationship to PDES. To our knowledge, no publications dealing with the relationship between DES and PDES appeared after 1999.

7.5.1 PDES self assessment

This subsection summarizes the views and opinions from inside of the PDES community to the situation in the PDES research field. In the lead article of the ORSA JoC series (Fujimoto, 1993b), R. Fujimoto asked the question, "*Parallel discrete event simulation: will the field survive?* " The question was motivated by the observed lack of impact of PDES techniques within the broader DES community. Fujimoto suggested that PDES techniques were too difficult to apply and too inscrutable to the non-parallel programmer. If a sequential program could be developed in a few hours and then run in a few hours more, why spend a day (or more) developing a parallel simulation that ran in a few minutes? He believed, however, that eventually performance demands would necessitate the adoption of PDES techniques by the simulation community at large. He suggested that the future of simulation as a technique might, in fact, hinge on the success of PDES, since PDES offered the best hope to modeling large, complex systems. Realizing the value of meeting the broader community half way, though, Fujimoto proposed several directions that could improve the usability of PDES. They are as follows:

- *Application specific libraries.* In certain application domains, a library of well-defined model objects or modules can be created. The user can build a simulation model using these modules which are appropriate for parallel simulation. The modules are ready to use, and the user does not need to create them. In this way, he/she is relieved of being concerned with PDES details. Examples of such application areas include telecommunication networks and digital logic simulations.
- *Parallel simulation languages.* Simulation languages similar to the traditional sequential languages like Simscript or Simula, but suitable for parallel execution could help wider adoption of PDES. Important question in their design is the degree of transparency. A totally transparent language is ideal for use, but could

easily lead to ineffective execution. A related question is what are the appropriate language constructs which are convenient for use and at the same time allow efficient execution.

- *Support for shared state.* Many simulation models are naturally programmed using variables shared among multiple simulation objects. While this presents no problem for sequential execution using common event list, sharing variables among LPs with different simulation times, and across processor boundaries is more difficult, should it be time effective. Excessive use of query events for inquiring state of another LP leads usually to performance degradation. Efficient solution provides the so called push-processing, i.e. transmitting state information before it is needed, or the space-time memory (Ghosh & Fujimoto, 1991).
- *Automatic parallelization.* Extant parallel simulators usually do not allow to use models created for sequential simulation; manual rewriting is necessary. Automatic parallelization of such model is a way to overcome this problem. Fujimoto outlines how it can be done using the abstraction of space-time memory, and discusses its performance.

The responses to Fujimoto's observations were somewhat mixed. Reynolds suggested that any focus on usability, while perhaps well-intentioned, might serve only as a distraction to the PDES research community. Reynolds asserted that the key issue to acceptance of parallel simulation is the performance. If it will be fast enough, it will be accepted (Reynolds, 1993).

In their response, Unger and Cleary echoed Fujimoto's call for an investment in tools, specifically, tools that support the incremental performance improvement of parallel simulations (Unger & Cleary, 1993). Bagrodia also concurred with Fujimoto's observations and stressed his belief in the central role that parallel simulation languages must play in the widespread adoption of PDES techniques (Bagrodia, 1993). Jason Lin discussed the importance of performance predictors and the need for parallel simulation engines of the future to be modularized and tailorable (Lin, 1993). Abrams observed what he referred to as a "culture clash" between parallel simulationists and modeling methodologists (Abrams, 1993). This served as a basis for more detailed comparison of the attitudes of these two camps in (Page & Nance, 1994).

A few years later, Nicol provided his view of the poor acceptance of PDES (Nicol, 1997a). His observations differ from the previous ones. He noted that engineers find the PDES community unconvincing since many of the problems studied in PDES can be adequately solved using sequential processors, and the applicability of PDES solutions to large "industrial sized" applications is not completely evident. Nicol also asserted that with the exception of communications networks, the applications studied in the PDES domain are not highly relevant. Scientists find PDES research unconvincing due to haphazard and flawed experimental design. Modelers find PDES too complicated. He observed that performance is a constraint, not an objective function and that factors such as model reusability and maintenance, model development, analysis of results, and interoperability are often more important than performance.

Adopting a Total Quality Management (TQM) approach, Nicol suggested that PDES should be assessed in terms of its customer base and their needs. He identified five classes of customer: industry, government, management, PDES colleagues, and graduate students. He contended that the PDES community served its management,

colleagues and graduate students well, but that industry and government were less well-served by the community. He suggested that the PDES community should expand its notion of relevant applications to include very large models on moderately parallel platforms and distributed platforms. Further, that the PDES community should emulate relevant application environments when evaluating its approaches, better analyze scalability, and make the language of PDES more user-friendly. Lastly, Nicol stressed that PDES researchers should strive to become better experimentalists.

7.5.2 Modeling methodological view

The objective of a simulation is to serve as an analysis and decision making tool. It is crucially important that the analysis results and the simulation-based decision be correct. Although the results are actually obtained from a model represented in the form of a computer program, simulation cannot be viewed as a mere exercise in programming. The programming language representation of a model is just one of the possible forms. This representation necessarily contains many implementation related details that obscure the clear expression of model behavior. Moreover, the use of a particular programming language has direct influences on the structure of the model formulation itself. These facts caused that a shift from a program-centric view of the simulation process to the model-centric view occurred in the late 1970's. Simulation modeling methodologies have been developed that focus on the question how a simulation model should be constructed in order to increase the likelihood of results and decision correctness.

Although the recommendations for model construction found in the literature differ in some details, they all stress that the model description should be natural. This allows for an agreement between the mental model formed in the mind of the modeler on one hand and its "materialized" description on the other hand. The necessity of such agreement is justified by the effort to minimize the risk of model errors. The conceptual frameworks introduced in section 7.1.2 provide a modeler with a means to construct a mental picture of the model. Then a model description language utilizing the selected conceptual framework enables to achieve the agreement between the mental picture and the model representation (Page & Nance, 1994).

In the context of a methodological framework, the development of a model consists of several stages. As the simulation study proceeds, the model formulation takes on several forms. In other words, the primary model representation is gradually translated until the final simulation program is created. The translation itself has to be verified and validated to ensure its correctness. Figure 7-13 shows a typical scenario of model development (Liu, 2003). A more detailed discussion can be found in most simulation textbooks, e.g. (Law & Kelton, 2000). It should be noted that the development process may not be sequential – it usually requires feedback to previous steps for further refinement. The steps are as follows:

- *Objectives.* This step is to formulate the problem. The goals and objectives should be clearly identified before we set out to develop the simulation model. Knowing the modeled real system itself is not sufficient, since the model formulation depends on the goals of the simulation.
- *Conceptual Model.* A conceptual model includes identification of system state variables and their evolution over time. The discussion about state variables is included in section 7.1.1.

- *Mathematical Model.* The mathematical model, also called algorithmic model, is developed from the conceptual model. An important aspect of this step involves statistical analysis to provide proper probabilistic input model to drive the simulation.
- *Computational model.* The mathematical model needs to be translated into a computer-recognizable format by using either a high-level simulation language or a generic computer programming language, possibly utilizing a simulation library. That is, the model must be implemented as a computer program.
- *Verification.* The computational model must be verified whether it is consistent with the mathematical or the algorithmic model. In other words, the program should run properly and generate results consistent with the algorithmic specification.
- *Validation.* The task of validation is to determine whether the model is an accurate and appropriate representation of the real system, in the context of simulation objectives (step 1). This is necessary since otherwise the answers obtained through simulation will not be valid, and the whole effort wasteful.

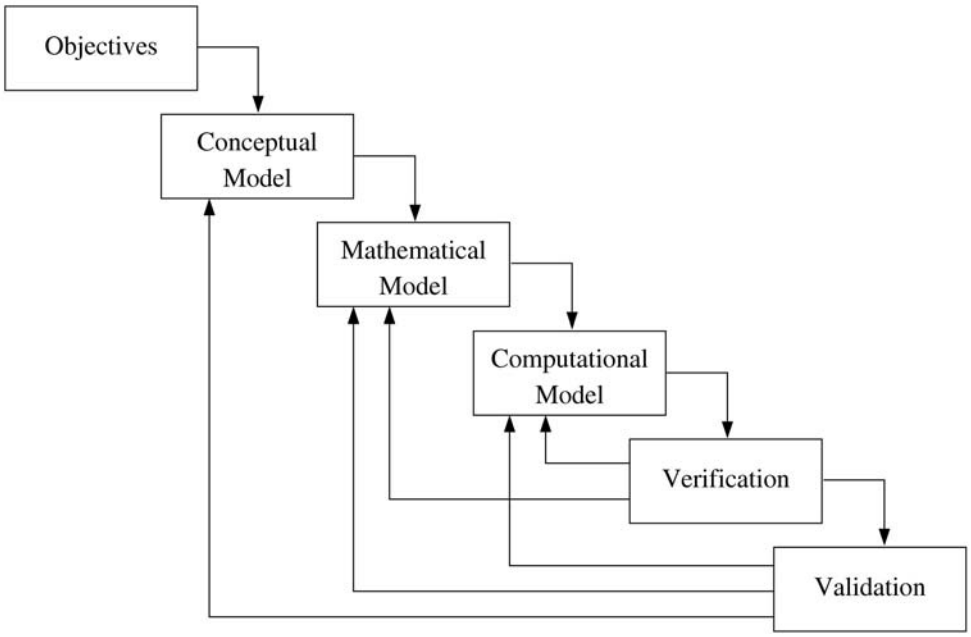


Figure 7-13. Simulation model development steps.

Unfortunately, the methodology and conceptual frameworks did not get appropriate recognition in PDES. A typical PDES study views the modeled system as a collection of physical processes which are simulated by a corresponding collection of logical processes (or submodels) interacting through message passing. This is a different language using terms unknown in the general simulation community, and ignoring the standard conceptual frameworks with their role in model construction. Furthermore, little distinction between model and simulation program is made. In fact, the simulation

program is usually the only model representation. Thus the program-centric approach is followed, and the model-centric view which is the result of a long-time research in modeling methodology is here not taken into account. Another flaw is that without utilizing the conceptual frameworks, the program is often unnatural representation of the modeled system. Further deviation from a natural representation may be forced by execution requirements and performance tuning. These discrepancies in model representation and the lack of other model representations appear as obstacles in model verification and validation, making an assessment of simulation correctness even more difficult.

7.5.3 User transparency and PDES

On one hand, commercial sequential simulation languages and environments provide the modeler with a convenient GUI for model creation and execution, often including also support for results analysis. On the other hand, extant parallel simulators provide a much lower level interface. Moreover, parallel model execution is in general not possible in a user transparent way, i.e. without the user being concerned with problems internal to parallel simulation, such as synchronization. This not only requires the user to be closely familiar with these issues. The effort needed to solve these extra issues dissipates the user concentration on simulation objectives themselves which may contribute to model errors. The current status of user transparency in PDES is summarized in the following paragraphs.

In conservative methods, good lookahead is crucial for acceptable performance. It must be extracted and explicitly specified for the model. Although there was a singular attempt to automate this task (Cota & Sargent, 1990), no general method exists, and this burden is usually up to the modeler. There is one special case known as an exception from the above general statement. It is the approach used in conservative simulators of large communication networks, such as Nops (Poplawski & Nicol, 1998), or DaSSF (Liu & Nicol, 2001). Here the model consists of network components interconnected through communication channels. The source of lookahead is the non-zero minimum delay a message sent between a pair of components experiences in a channel. The network components themselves have no lookahead. The minimum delays of channels are part of model description, and are supplied by the user. It is important that the user is not expected to have any PDES-specific knowledge to supply this information.

The Time Warp optimistic approach does not need explicit specification of lookahead. The essence of the method allows exploit the available parallelism. However, it suffers from other problem caused by allowing LPs to send and process risk messages (see section 7.3.3). This can lead to processing messages that are inconsistent with the state of the LP. Such messages are destined to be canceled by rollback, but they can produce non-sensical data and actions which have the ability to corrupt the operating environment in a way non-recoverable by any rollback (Nicol, 1997b). As a result, the program can crash, or behave incorrectly giving wrong results without any hint of error.

To clarify the relationship of this problem to PDES transparency, let us quote from (Nicol, 1997b): "*Time Warp has long been advertised as making synchronization transparent to the user; it most definitely is not transparent if in order to ensure that it runs without crashing we must augment the code with extensive consistency checks.*" Determining consistency of a message may be a very difficult task. However, to arrive

at a correct solution, it is necessary to program the simulation with this issue in mind. This again enforces the user to think differently about the model due to its parallel execution, pushing parallel simulation far from being user transparent.

7.6 High level architecture

An important part of todays PDES stuff is embodied by the High Level Architecture (HLA). This is a general purpose architecture allowing simulation re-use and interoperability. It was developed under the leadership of US Defense Modeling and Simulation Office (DMSO). It was approved as an open standard through IEEE as Standard 1516 in September 2000. All simulations developed for US DoD are required to comply to this standard.

In the context of HLA, a simulation consists of a collection of cooperating simulators. Each individual simulator is referred to as a *federate*. The collection of federates is called a *federation*. The HLA is defined by the following three ingredients:

1. HLA rules, that define the design principles used in the HLA.
2. The Object Model Template (OMT) that provides a standard format for describing information which is of interest to more than one federate.
3. The Runtime Infrastructure (RTI) that serves to coordinate the execution of federates and exchange information among them.

Next we describe each of these ingredients in more detail (Fujimoto, 2000).

7.6.1 HLA rules

The rules formulate general requirements imposed on the federation and individual federates, as follows:

1. Federations shall have an HLA Federation Object Model (FOM) documented in accordance with the HLA Object Model Template.
2. In a federation, all simulation-associated object instance representation shall be in the federates, not in the RTI.
3. During a federation execution, all exchange of FOM data among federates shall occur via the RTI.
4. During a federation execution, federates shall interact with the RTI in accordance with the HLA Interface Specification
5. During a federation execution, an attribute of an instance of an object shall be owned by at most one federate at any time.
6. Federates shall have an HLA Simulation Object Model (SOM), documented in accordance with the HLA OMT.
7. Federates shall be able to update and/or reflect any attributes of objects in their SOMs and send and/or receive SOM interactions externally as specified in their SOMs.
8. Federates shall be able to transfer and/or accept ownership of attributes dynamically during a federation execution, as specified in their SOMs.

9. Federates shall be able to vary the conditions (e.g. thresholds) under which they provide updates of attributes of objects, as specified in their SOMs.
10. Federates shall be able to manage local time in a way that will allow them to coordinate data exchange with other members of a federation.

7.6.2 Object model template

The object model template (OMT) refers to a set of tables that provide a means for documenting object models in the HLA. The *simulation object model* (SOM) specifies the object attributes a simulator can update and/or receive. The common objects used by simulators participating in a federation are specified in the *federation object model* (FOM).

The OMT consists of the following tables:

1. The *object model identification table* provides general information about the FOM or SOM such as its name, purpose, version, and point of contact information.
2. The *object class structure table* specifies the class hierarchy of objects in the SOM/FOM.
3. The *attribute table* enumerates the type and characteristics of object attributes.
4. The *interaction class structure table* defines the class hierarchy of the interactions in the SOM/FOM.
5. The *parameter table* defines types and characteristics of interaction parameters.
6. The *routing space table* specifies the nature and meaning of routing spaces that are used to efficiently distribute data throughout large federations. This is related to the data distribution management in the RTI.
7. The *FOM/SOM lexicon* is used to describe the meaning of all terms used in the other tables, such as meaning of object classes, attributes, interaction classes, and parameters.

7.6.3 Runtime infrastructure

Runtime infrastructure (RTI) provides services for federates cooperation. Federates, unlike submodels in "classical" distributed simulators, contain beside the simulation model also the local simulation executive. The local executives of individual federates cooperate through RTI.

To achieve the generality of the RTI, there is a clear separation of functionality between the federates and the RTI, where all simulated-system-dependent functionality is encapsulated within the federates. This separation is enforced also by the HLA interface specification which defines a set of services provided by federates or by the RTI. The services fall into the following categories:

- *Federation management* services allow to create and delete federation executions, allow simulations to join or resign from existing federations, and to pause, checkpoint and resume a federation execution.
- *Declaration management* services provide the means for simulations to establish their intent to publish object attributes and interactions, and to subscribe to updates and interactions produced by other federates.

- *Object management* services allow simulations to create and delete object instances, and to produce and receive individual attribute updates and interactions.
- *Ownership management* services enable the transfer of ownership of object attributes during the federation execution.
- *Time management* services coordinate the advancement of simulation time, and its relationship to wallclock time during the federation execution.
- *Data distribution management* services control the distribution of state updates and interactions among federates. The goal is that each federate receives all of the information relevant to it, and no other information.

Because our interest is oriented towards virtual time related problems, we describe in more detail the time management services.

7.6.4 Time management and the lookahead

The messages sent among federates are transferred through RTI. Because of the generality of HLA, the messages fall into two categories: *timestamp order* (TSO) messages which carry the simulation time timestamp, and *receive order* (RO) messages with no timestamp. For TSO messages, the RTI ensures their delivery to each particular destination federate in timestamp order. In other words, a TSO message is delivered to its destination federate only if there will be no messages with smaller timestamp. If delivering a message could violate the timestamp order in the destination federate, the message is temporarily kept within RTI. On the other hand, the RO messages are not delayed in RTI due to their ordering, and are delivered immediately. Thus, RO messages are of no interest from the synchronization point of view. For the sake of generality again, each federate can be categorized as *time regulating*, and/or *time constrained*, according to its two corresponding boolean flags. A time regulating federate can generate TSO messages, and a time constrained federate can receive TSO messages.

An event driven federate must process all its events in timestamp order. For a time constrained federate, these events include internal events and also events generated by other federates and received as TSO messages. It is the responsibility of RTI not to sent any TSO message with time stamp less than the local simulation time of the destination federate. To achieve this, local time advancement of the federate must be negotiated with the RTI. The negotiation consists of three steps. First, the federate invokes a time management service to request its local time to advance, e.g. to the timestamp of the next local event to be processed. Next, the RTI delivers some number (possibly zero) of messages to the federate. These are the messages which are safe to deliver at the moment. Then the RTI completes the negotiation by granting the time advance, possibly to different value than the one requested by the federate. The federate can proceed up to the point of the granted simulation time. Then a new advancement must be negotiated with the RTI.

Synchronization of simulation times of federates, as outlined, is controlled by RTI using a conservative approach. The key to implementing the time management services within the RTI is to compute the *lower bound on timestamp* (LBTS) for each federate (cf. formula 4). For federate i, the value of $LBTS_i$ provides a lower bound on timestamp of messages that may be received by federate i in the future. Once $LBTS_i$ has been computed, the RTI can deliver to federate i all TSO messages containing timestamp

less than $LBTS_i$. Further, the RTI prevents the federate from advancing its local time beyond $LBTS_i$, so that TSO order of message delivery to the federate can be guaranteed.

The LBTS value can be computed similarly to formula (4). To compute it, the RTI must consider the smallest timestamp of any TSO message a federate can send. The local time of the federate provides a bound of such timestamp. Also the timestamps of messages within the RTI and in the interconnection network must be considered. Further, the lookahead of federates is needed in the computation.

In HLA, each federate supplies a single lookahead value L. The semantics of the value is that the federate shall not send a message with timestamp less than its local simulation time plus the lookahead L (Fujimoto, 1998). According to the classification presented in subsection 7.4.3, this is a bounded time lookahead corresponding to the minimum timestamp increment in the federate. The lookahead of a federate can change dynamically during the simulation. However, lookahead cannot be instantaneously reduced. If a federate requires to reduce its lookahead by K units of time, its local time must advance K units before this change can take effect. If lookahead was reduced instantaneously, the RTI could not guarantee the timestamp order of message delivery of TSO messages.

Federates with zero lookahead are allowed in HLA. The RTI interface contains special functions for that case. Details can be found e.g. in (Fujimoto, 1997).

7.7 Conclusion

We have presented a survey of concepts, techniques, and topics constituting the core of the research and engineering area of Parallel Discrete Event Simulation. Of course, not all methods or approaches to individual problems can be captured in such survey, nor every single problem arising in this area can be mentioned. In spite of that, we believe that the document gives a reasonable overall insight into the area as a whole, and, at the same time, a more detailed description of some issues which draw the most of our attention.

References

ABRAMS, M. (1993). Parallel discrete event simulation: Fact or fiction. In: *ORSA Journal on Computing*, Vol. 5, No. 3, pp. 231-233.

ANDRADOTTIR, S. – OTT, T. (1995). Time–segmentation parallel simulation of network of queues with loss or communication blocking. In: *ACM Transactions on Modeling and Computer Simulation*, Vol. 5, No. 4, pp. 269-305.

AYANI, R. – RAJAEI, H. (1992). Parallel simulation using conservative time windows. In: *Proceedings of the 24th Winter Simulation Conference*, ACM Press, pp. 709-717.

BAGRODIA, R. (1993). A survival guide for parallel simulation. In: *ORSA Journal on Computing*, Vol. 5, No. 3, pp. 234-235.

BALCI, O. (1988). The implementation of four conceptual frameworks for simulation modeling in high-level languages. In: *Proceedings of the 20th Winter Simulation Conference*, ACM Press, pp. 287-295,

BOUKERCHE, A. – TROPPER, C. (1995). SGTNE: Semi-global time of the next event algorithm. In: *Proceedings of the 9th Workshop on Parallel and Distributed Simulation*, IEEE CS Press, pp. 68-77.

BRYANT, R. E. (1977). Simulation of packet communication architecture computer systems. *Technical Report MIT-LCS-TR-188*, M.I.T.

CAI, W. – TURNER, S. (1990). An algorithm for distributed discrete-event simulation – the "carrier null message" approach. In: *Proceedings of the SCS Multiconference on Distributed Simulation*, pp. 3-8.

CAROTHERS, C. D. – PERUMALLA, K.S. – FUJIMOTO, R.M. (1999). Efficient optimistic parallel simulations using reverse computation. In: *ACM Transactions on Modeling and Computer Simulation*, Vol. 9, No. 3, pp. 224-253.

CHANDY, K. M. – MISRA, J. (1979). Distributed simulation: A case study in design and verification of distributed programs. In: *IEEE Transactions on Software Engineering*, Vol. 5, No. 5, pp. 440-452.

CHANDY, K. M. – MISRA, J. (1981). Asynchronous distributed simulation via a sequence of parallel computations. In: *Communications of the ACM*, Vol. 24, No. 11, pp. 198-205.

CHANDY, K. M. – SHERMAN, R. (1989a). The conditional event approach to distributed simulation. In: *Proceedings of the SCS Multiconference on Distributed Simulation*, Vol. 21, pp. 93-99.

CHANDY, K. M. – SHERMAN, R. (1989b). Space, time, and simulation. In: *Proceedings of the SCS Multiconference on Distributed Simulation*, Vol. 21, pp. 53-57.

COMFORT, J. – YAO, W. – LI, Q. (1984). The design of a multi-microprocessor based simulation computer – III. In: *Proceedings of the 17th Annual Simulation Symposium*, pp. 227-241.

COTA, B. A. – SARGENT, R. G. (1990). A framework for automatic lookahead computation in conservative distributed simulations. In: *Proceedings of the SCS Multiconference on Distributed Simulation*, Vol. 22, pp. 56-59.

COWIE, J. – NICOL, D. M. – OGIELSKI, A. T. (1999). Modeling the global internet. In: *Computing in Science & Engineering*, Vol. 1, No. 1, pp. 42-50.

FERSCHA, A. – TRIPATHI, S. K. (1994). Parallel and distributed simulation of discrete event systems. *Technical Report CS-TR-3336,* University of Maryland at College Park.

FUJIMOTO, R. M. (1990a). Parallel discrete event simulation. In: *Communications of the ACM*, Vol. 33, No. 10, pp. 30-53.

FUJIMOTO, R. M. (1990b). Performance of Time Warp under synthetic workloads. In: *Proceedings of the SCS Multiconference on Distributed Simulation*, Vol. 22, pp. 23-28.

FUJIMOTO, R. M. (1993a). Future directions in parallel simulation research. In: *ORSA Journal on Computing*, Vol. 5, No. 3, pp. 245-248.

FUJIMOTO, R. M. (1993b). Parallel discrete event simulation: Will the field survive? In: *ORSA Journal on Computing*, Vol. 5, No. 3, pp. 213-230.

FUJIMOTO, R. M. (1996). Ten years of PADS: Where we've been, where we're going. In: *Proceedings of the 10th Workshop on Parallel and Distributed Simulation*, IEEE CS Press.

FUJIMOTO, R. M. (1997). Zero lookahead and repeatability in the High Level Architecture. In: *Proceedings of the 1997 Spring Simulation Interoperability Workshop.*

FUJIMOTO, R. M. (1998). Time management in the High Level Architecture. In: *Simulation*, Vol. 71, No. 6, pp. 388-400.

FUJIMOTO, R. M. (2000). *Parallel and Distribution Simulation Systems*. Wiley-Interscience, pp. 320.

GHOSH, K. – FUJIMOTO, R. M. (1991). Parallel discrete event simulation using space-time memory. In: *Proceedings of the 1991 International Conference on Parallel Processing*, Vol. III, Algorithms & Applications, pp. 201-208.

GREENBERG, A. G. – LUBACHEVSKY, B. D. – MITRANI, I. (1991). Algorithms for unboundedly parallel simulations. In: *ACM Transactions on Computer Systems*, Vol. 9, No. 3, pp. 201-221.

GROSELJ, B. – TROPPER, C. (1988). The time of next event algorithm. In: *Proceedings of the SCS Multiconference on Distributed Simulation*, Vol. 19, pp. 25-29.

JEFFERSON, D. (1985). Virtual time. In: *ACM Transactions on Programming Languages and Systems*, Vol. 7, No. 3, pp. 404-425.

JONES, K. G. – DAS, S. R. (2001). Time-parallel algorithms for simulation of multiple access protocols. In: *Proceedings of the 9th International Symposium in Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, IEEE CS Press, pp. 49-58.

KNUTH, D. E. – MCNELEY, J. L. (1964). SOL – A symbolic language for general–purpose systems simulation. In: *IEEE Transactions on Electronic Computers*, Vol. 13, No. 4, pp. 401-408.

LAMPORT, L. (1978). Time, clocks, and the ordering of events in a distributed system. In: *Communications of the ACM*, Vol. 21, No. 7, pp. 558-565.

LAW, A. M. – KELTON, W. D. (2000). *Simulation Modeling and Analysis*. McGraw-Hill Education.

LIN, Y. B. (1993). Will parallel simulation research survive? In: *ORSA Journal on Computing*, Vol. 5, No. 3, pp. 236-238.

LIN, Y. B. – LAZOWSKA, E. D. (1991). A time-division algorithm for parallel simulation. In: *ACM Transactions on Modeling and Computer Simulation*, Vol. 1, No. 1, pp. 73-83.

LIU, J. (2003). *Improvements in Conservative Parallel Simulation of Large Scale Models*. PhD thesis, Department of Computer Science, Dartmouth College, Hanover, New Hampshire, USA.

LIU, J. – NICOL, D. M. (2001). *DaSSF 3.1 User's Manual*. Department of Computer Science Dartmouth College, USA.

LOW, Y. H. – LIM, C. C. – CAI, W. ET AL. (1999). Survey of languages and runtime systems for parallel discrete event simulation. In: *Simulation*, Vol. 72, No. 3, pp. 170-196.

LUBACHEVSKY, B. D. (1989). Efficient distributed event-driven simulations of multiple-loop networks. In: *Communications of the ACM*, Vol. 32, No. 1, pp. 111-123.

MEŇHART, P. (1995). *Distribuovaný simulačný systém a tvorba distribuovaného modelu*. PhD thesis, KIVT FEI Slovenská Technická Univerzita, Bratislava.

MEYER, R. A. – BARGRODIA, R. L. (1999). Path lookahead: a data flow view of PDES models. In: *Proceedings of the 13th Workshop on Parallel and Distributed Simulation*, IEEE CS Press, pp. 12-19.

MISRA, J. (1986). Distributed discrete event simulation. In: *ACM Computing Surveys*, Vol. 18, No. 1, pp. 39-65.

NEUSCHL, Š. ET AL. (1988). *Modelovanie a simulácia*. Alfa, Bratislava.

NICOL, D. M. (1993). The cost of conservative synchronization in parallel discrete event simulations. In: *Journal of the ACM*, Vol. 40, No. 2, pp. 304-333.

NICOL, D. M. (1996). Principles of conservative parallel simulation. In: *Proceedings of the 28th Winter Simulation Conference*, pp. 128-135.

NICOL, D. M. (1997a). Parallel discrete event simulation: so who cares. In: *Proceedings of the 11th Workshop on Parallel and Distributed Simulation*, IEEE CS Press.

NICOL, D. M. (1998). Scalability, locality, partitioning and synchronization PDES. In: *Proceedings of the 12th Workshop on Parallel and Distributed Simulation*, IEEE CS Press, pp. 5-11.

NICOL, D. M. – GREENBERG, A. G. – LUBACHEVSKY, B. D. (1994). Massively parallel algorithms for trace-driven cache simulations. In: *IEEE Transactions on Parallel and Distributed Systems*, Vol. 5, No. 8, pp. 849-859.

NICOL, D. M. – LIU, X. (1997b). The dark side of risk (what your mother never told you about Time Warp). In: *Proceedings of the 11th Workshop on Parallel and Distributed Simulation*, pp. 188-195.

NICOL, D. M. – MICHEAL, C. C. – INOUYE, P. (1989). Efficient aggregation of multiple LPs in distributed memory parallel simulation. In: *Proceedings of the 21th Winter Simulation Conference*, pp. 680-685.

PAGE, E. H. (1999). Beyond speedup: PADS, the HLA and Web-based simulation. In: *Proceedings of the 13th Workshop on Parallel and Distributed Simulation*, IEEE CS Press, pp. 2-9.

PAGE, E. H. – NANCE, R. E. (1994). Parallel discrete event simulation: A modeling methodological perspective. In: *Proceedings of the 8th Workshop on Parallel and Distributed Simulation*, pp. 88-93.

PERUMALLA, K. S. (2005). "μsik" A micro-kernel for parallel/distributed simulation systems. In: *Proceedings of the 19th Workshop on Principles of Advanced and Distributed Simulation*, IEEE CS Press, pp. 59-68.

POPLAWSKI, A. L. – NICOL, D. M. (1998). Nops: A conservative simulation engine for TeD. In: *Proceedings of the 12th Workshop on Parallel and Distributed Simulation*, pp. 180-187.

PREISS, B. R. – LOUCKS, W. M. (1990). The impact of lookahead on the performance of conservative distributed simulation. In: *Proceedings of the 1990 European*

Multiconference—Simulation Methodologies, Languages and Architectures, SCS, pp. 204-209.

REYNOLDS, P. F. (1988). A spectrum of options for parallel simulation. In: *Proceedings of the 20th Winter Simulation Conference*, pp. 325-332.

REYNOLDS, P. F. (1993). The silver bullet. *ORSA Journal on Computing*, Vol. 5, No. 3, pp. 239-241.

SOLČÁNY, V. – ŠAFAŘÍK, J. (1998). Improving accuracy of time-parallel approximate simulations of queueing networks. In: *Proceedings of the European Simulation Symposium*, pp. 26-28.

SOLČÁNY, V. – ŠAFAŘÍK, J. (2001). Parallel simulation for simulation practitioners. In: *Proceedings of the 35th Spring International Conference Modelling and Simulation of Systems MOSIS '01*.

SOLČÁNY, V. – ŠAFAŘÍK, J. (2003). Towards a user friendly parallel simulator of discrete event systems. In: *Proceedings of the 4th International Carpathian Control Conference (ICCC'2003)*.

STEINMAN, J. S. (1991). SPEEDES: Synchronous parallel environment for emulation and discrete event simulation. In: *Proceedings of the SCS Multiconference on Advances in Parallel and Distributed Simulation*, pp. 95-101.

SU, W. K. – SEITZ, C. L. (1989). Variants of the Chandy-Misra-Bryant distributed discrete-event simulation algorithm. In: *Proceedings of the SCS Multiconference on Distributed Simulation*, Vol. 21, pp. 38-43.

UNGER, B.W. – CLEARY, J. G. (1993). Practical parallel discrete event simulation. In: *ORSA Journal on Computing*, Vol. 5, No. 3, pp. 242-244.

ŠAFAŘÍK, J. (1983). *Príspevok k automatizácii tvorby simulačných programov*. PhD thesis, EF SVŠT, Bratislava.

WAGNER, D. – LAZOWSKA, E. (1989). Parallel simulation of queueing networks: Limitations and potentials. In: *Performance Evaluation Review*, Vol. 17, No. 2, pp. 146-155.

WANG, J. J. – ABRAMS, M. (1993). Determining initial states for time-parallel simulations. In: *Proceedings of the 7th Workshop on Parallel and Distributed Simulation*, pp. 19-26.

WANG, J. J. – ABRAMS, M. (1995). The impact of lookahead on conservative simulation. *Technical Report ncstrl.vatech.cs//TR-95-03*, Virginia Polytechnic Institute and State University, Blacksburg, VA, USA.

YÜCESAN, E. – SCHRUBEN, L.W. (1992). Structural and behavioral equivalence of simulation models. In: *ACM Transactions on Modeling and Computer Simulation*, Vol. 2, No. 1, pp. 82-103.

ZEIGLER, B.P. – KIM, T.G. – PRAEHOFER, H. (2000). *Theory of Modeling and Simulation*. 2nd edition. Academic Press, pp. 510.

8 COMMUNITY STRUCTURES IN NETWORKS

8.1 Models of networks

Modeling of networks is useful, because models present a motivation of their growth's dynamics. The produced graphs possess some computable and measurable properties as a result of the primary motivation. If similar property is measured in a real network, theoretical models may give us the answer why the measured property appeared in the network. The correlation between vertices of similar degree was the result of higher probability of the older vertices to get connected in this model. Network models are the formalizations of our perception of networks.

8.1.1 Small-world networks

Several observed networks attain the so called *small-world effect*. In these networks the average vertex-to-vertex distances increase only logarithmically by the increase of the total number of vertices. However, short geodesic distance is also present in random networks. But because of the smallness of the networks also edge transitivity is present in most of the natural networks. The random networks miss this property. We characterize two models for small-world effect modeling. Both of these models are more precisely described in (Watts, J. D., 2003).

The α-model. The process of graph generation of the α-model starts with a graph of n vertices without any edges. The edges are added in successive steps until they reach the number of $m = kn/2$, where k is the mean vertex degree.

The procedure for addition of an edge in α-model is depicted on Figure 8-2. It uses the parameters $R_{i.j}$ is the likelihood that i gets connected to j, $m_{i,j}$ is the number of vertices adjacent to both i and j, k is the mean degree of the vertices, $p \in [0,1]$ tunable baseline, $\alpha \geq 0$ is tunable parameter.

The edges are added in cycles. The addition of a new edge is repeated on every vertex in random order with the restriction that in every cycle every vertex is processed exactly once. k cycles are proceeded to generate the graph.

Watts-Strogatz model. The model of Watts and Strogatz (Watts, J. D., 2003; Newman, 2000) presents the idea how a highly clustered network with high geodesic distance become a "small-world". The process starts with a regular lattice graph. The graph has

the geodesic distance linearly depending on the size of the graph, by rewiring only a small portion of edges; the average distance rapidly decreases and becomes $l \approx \log N$. The rewiring is shown on Figure 8-1.

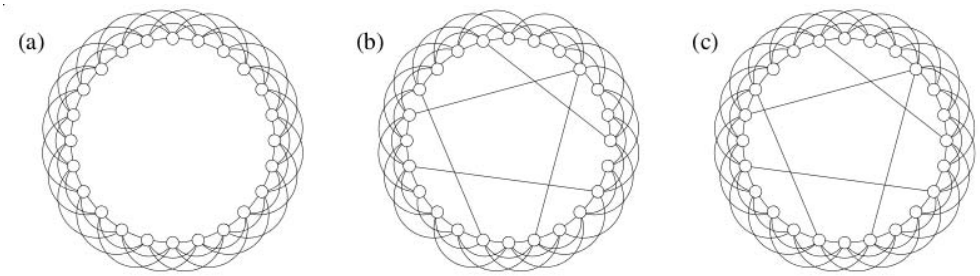


Figure 8-1. Watts-Strogatz small world model. (a) One-dimensional lattice with n vertices is created. Every vertex is connected to k vertices on both sides (b) Portion of p edges of the lattice are rewired randomly to uniformly chosen vertex of the graph (c) Variation of the edge rewiring, p edges are added connecting two randomly chosen vertices.

1. Choose a vertex i;
2. forall vertex j do
3. if vertices i and j are connected then
4. $R_{i,j} = 0$;
5. else
6. $$R_{i,j} = \begin{cases} 1 & m_{i,j} \geq k \\ \left\lfloor \frac{m_{i,j}}{k} \right\rfloor^{\alpha} (1-p) + p & k > m_{i,j} > 0 \\ p & m_{i,j} = 0 \end{cases};$$
7. normalise R to obtain probabilities of connection R to the other vertices: $P_{i,j} = \frac{R_{i,j}}{\sum_{l \neq j} R_{i,l}}$;
8. choose vertex j^* from uniform random variable $P_{i,j}$;
9. connect i to j^*;

Figure 8-2. Addition of a new edge to vertex i. The vertex chosen to be connected to i is selected based on the existing number of adjacencies to vertices i and j represented in $m_{k,l}$.

Figure 8-3 shows the change of the clustering and the geodesic distance depending on the parameter p. The interval of possible values for p where the graph possesses both small geodesic distance and high clustering coefficient is the region of small-world graphs.

Watts-Strogatz model introduces two extremes: the ordered lattice graphs and random networks. Lattice graphs ($p = 0$) exhibit high clustering coefficient. On the other hand, their geodesic distance is high. Random networks ($p = 1$) have low clustering coefficient and small distances.

The geodesic distance l in small-world networks have been found to depend logarithmically on the number of the vertices:

$$l = \log(N)$$

Logarithmic dependence allows the average distance between the vertices to be small even in large networks.

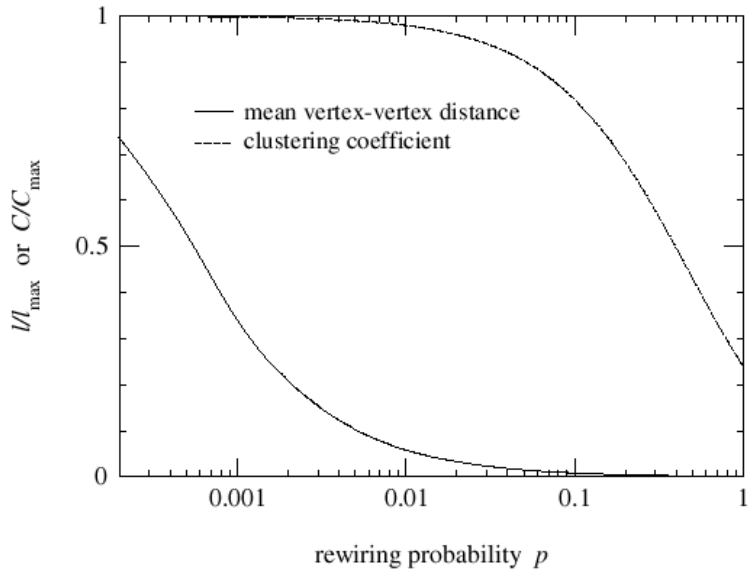


Figure 8-3. Watts-Strogatz model, distance and clustering coefficient. By increasing the rewiring probability p in the Watts-Strogatz model (Watts & Strogatz, 1998) the clustering coefficient decreases slower than the average vertex to vertex distance. Therefore a region possessing both high clustering coefficient and low geodesic distance appears.

8.1.2 Expanding networks

In absence of data on real networks the assumption of Erdős and Rényi was that the complex networks are random and the vertices' degree follows the Poisson distribution (Erdős & Rényi, 1960). By the expansion of the Internet it was revealed that the big and evolving networks such as the Web pose high degree of self-organization and they are likely to be much different from the random networks. The degree distribution appears to follow the power-law distribution for a lot of natural networks. The core reason of the $O(k^{-\alpha})$ degree distribution is concealed in the growth of the network and in the preferential attachment of the new incoming vertices (Albert et al., 1999).

Citation network

Perhaps the first known paper discussing the power-law degree distribution of real networks was introduced by Price (Newman, 2003c). Price discusses the network of scientific citations. Every paper refers to others in its bibliography. Therefore the edges of the network are directed. As all of the papers refer only to already published ones, the network does not contain directed cycles (only with rare exceptions).

The new papers entering the citation network are supposed to cite papers present in the scientific community preferentially; those which are already cited by more papers are likely to be popular and are being cited again by higher probability.

Let p_k be the fraction of vertices with in-degree k, clearly $\sum_k p_k = 1$ holds. The model sets a constant mean out-degree of the vertices denoted as m. The in-degree varies from vertex to another. The mean in-degree and the out-degree equals, so that $\sum_k kp_k = m$, where m is a parameter of the model, it can take any even non-integer value.

The process of creation of the graph is by adding new vertices, attaching them to m old vertices with probability proportional to their in-degree. As the in-degree of the new-vertices is zero some minimum probability for attachment has to be assigned. The probability of attachment of a new vertex to any old vertex is proportional to $k+1$. Thus the probability of attaching a vertex of degree k is

$$\frac{(k+1)p_k}{\sum_k (k+1)p_k} = \frac{(k+1)p_k}{m+1}$$

From this follows that the mean number of connections to vertices with in-degree k is $m(k+1)p_k / m+1$. Addition of incoming link to vertex with in-degree k causes the decrease of the number of vertices with in-degree k. On the other hand, vertices with in-degree $k-1$ for $k>0$ flow in. This leads to the change of the number of vertices with in-degree k (for n vertices this is np_k). In the following equations let $p_{k,n}$ denote the value p_k when graph has n vertices.

$$(n+1)p_{k,n+1} - np_{k,n} = \left[kp_{k-1,n} - (k+1)p_{k,n}\right]\frac{m}{m+1}$$

For $k=0$ it is

$$(n+1)p_{0,n+1} - np_{0,n} = 1 - p_{0,n}\frac{m}{m+1}$$

The solution degree distribution is $p_k \approx k^{-(2+1/m)}$, hence it has a power-law distribution.

The Barabási-Albert model

The Barabási-Albert (BA) (Albert et al., 1999) model is similar, but simpler than the Price's model. The Barabási-Albert model operates on undirected graph instead of

directed graph as in the citation network. This significant simplification automatically resolved the problem of getting the first acquaintance in the model of Price. Although the BA approach might seem to be farther from the reality, the model miss such straightforward motivation as the citation network, the defined model poses to be more rigorous than Price's model and serves much better basis for further improvements.

The BA model considers growing network, in every step one vertex is added. The new vertex gets connected to m already present nodes with probability $(1-p)k_i + p$, where k_i is the number of the node's neighbors, p is the baseline parameter, the lower it is, the more preferential the connections become, higher p means preferring random connections. The graph generation starts with $N > m$ unconnected nodes. Clearly nodes with higher number of neighbors are more likely to receive new neighbors. This rich-gets-richer property of the graph evolution exploits some nodes with high number of connections and lot of nodes with few neighbors.

The probability of attaching old vertex with degree k is

$$\frac{kp_k}{\sum_k kp_k} = \frac{kp_k}{2m}$$

The number of vertices that increased their degree from k to $k+1$ is $mkp_k / 2m = kp_k / 2$ and is independent on m. The influx of vertices that increased their degree from $k-1$ to k is $kp_{k-1}/2$. Thus the number of vertices with degree k (this number equals to np_k) changes as

$$(n+1)p_{k,n+1} - np_{k,n} = \frac{1}{2}(k-1)p_{k-1,n} - \frac{1}{2}kp_{k,n}$$

if $k > m$, and for $k = m$ the influx of vertices with degree m is 1, thus the change of number of vertices degree m is

$$(n+1)p_{m,n+1} - np_{m,n} = 1 - \frac{1}{2}kp_{m,n}$$

The stationary solutions are found by setting $p_{k,n+1} = p_{k,n} = p_k$ and solving equation

$$p_k = \begin{cases} \frac{1}{2}(k-1)p_{k-1} - \frac{1}{2}kp_k & for\, k > m \\ 1 - \frac{1}{2}mp_m & for\, k = m \end{cases}$$

Solving the equation leads to value of p_k

$$p_k = \frac{2m(m+1)}{(k+2)(k+1)k} \approx k^{-3}$$

Fit-gets-rich model. The Barabási-Albert model of scale-free networks prefers the vertices present longer in the graph (Albert et al., 1999). The "rich-get-richer" and "first-mover-advantage" models partly contradict with the every day life experience. New companies emerge in the branch of the Web; become successful outstripping the others with longer tradition. The BA model of network expansion was extended by fitness attached to the vertices. This yields networks, which can be characterized as "first-mover-advantage", "fit-get-rich" or "winner-takes-all". The equilibrium of this competitive model was found similar to thermo dynamical properties of quantum gases (Bianconi & Barabási, 2001).

The probability of getting connected by the new node depends on number of the links k_i and on the fitness η_i:

$$\Pi_i = \frac{\eta_i k_i}{\sum_l \eta_l k_l}$$

Based on the distribution of fitness two states emerge:

Fit-get-rich. Different fitness values result few hubs with high number of connections following $P(k) \approx k^{-\gamma}$. If the fitness of the nodes is equal, the model reduces to the original BA model (Albert et al., 1999).

Winner-takes-all. In competition of links the vertices with the largest fitness receives all new connections and emerges as a clear winner. This phase predicts the "winner-takes-all" phenomenon.

Network as fractal

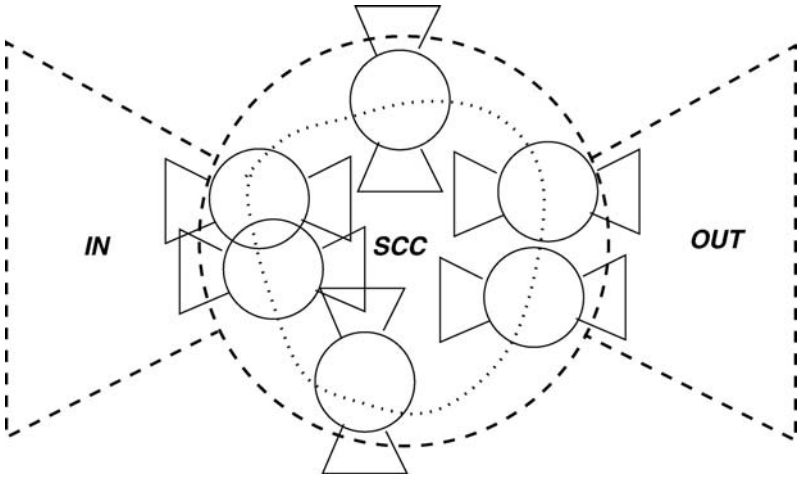


Figure 8-4. Illustration of bow-tie structure of the Web. The global properties can be found also in its parts. The Web exhibits such self-similarity.

The bow-tie structure of the Web was described by Broder et al. (Broder et al., 2000). Based on their study oriented networks can be decomposed to more parts:

- *Strongly connected component (SCC)* – The "core" of the network where the vertices are connected by directed paths.

- *IN* – Those vertices from which it is possible to reach the vertices of SCC, but unreachable from SCC.
- *OUT* – Vertices which are reachable from the SCC, but it is impossible to reach the SCC from the OUT.
- *Tendrils* – Paths not contained by the SCC leading out from the IN set, resp. into the OUT set.
- *Tubes* – Paths from IN to OUT.
- *Small components* – Behalf the giant component, several small components appear, with similar IN/SCC/OUT decomposition as in the giant component.

The model of Broder et al. was further elaborated by Dill et al. (Dill et al., 2002). Self-similarity of the Web had been found: specific parts of the network possess the same bow-tie structure as the whole network. See Figure 8-4.

8.2 Network clustering techniques

Data clustering is involved in the solution of many important problems. Clustering algorithms yield more clusters (groups) of instances from the data-set where each instance in a cluster is more similar to those inside the cluster than to instances outside the cluster. The considered similarity is depending from the domain.

Wide variety of problems ranging from identification of citation duplications, through marketing applications, such as customer profile clustering, seeking of DNA groups, etc. exist. Graph clustering is also exploited by several applications, such as social-network analysis, computer graphics, bioinformatics, and VLSI design. For identification of scientific publication duplications (Aleman-Meza et al., 2006) clustering is also a possible approach (McCallum et al., 2000).

The data-sets for clustering can differ in several ways. In the simplest case the instances of the data-set have the same attributes. However, they often differ in the number and type of attributes. Often the instances explicitly refer each other and thus create a network of instances. This network is exploited by graph-based partitioning algorithms. Graphs are appropriate for representation of elements in data-sets. These elements, depending on the domain being examined, might reference each other; the edges may be represented by weighted edges expressing similarity between the two elements. Graph partitioning algorithms can be thus often well applied for arbitrary set of instances, if the similarity between the instances were well estimated. We threat instances often as graph vertices in following text. It should be clear from the context, when we are considered about instance's attributes and when about its neighbors in the network of instances.

We recall some basic definition from the graph theory used is this chapter. Let $G(V,E)$ be an undirected graph. Let $E(X)$ denote the set of edges existing also in the graph connecting two different vertices in $\mathrm{X}:\mathrm{E}(\mathrm{X})=\{\{\mathrm{u},\mathrm{v}\}\in \mathrm{E} \mid \mathrm{u},\mathrm{v}\in \mathrm{X}\}$. *Subgraph induced* by a set $X \subseteq V$ is $G[X]=(X,E(X))$. Subgraph $G[X]$ is *clique* if $|E(X)|=C_2^{|X|}$. We denote degree of a vertex $\deg_G(u)=|\{u \mid \{u,v\}\in E\}|$. For a cut

S the degree is defined as $\deg_G(S) = \sum_{v \in S} \deg_G(v)$. Size of a cut S is $c_G(S) = \left|\{\{u,v\} \mid u \in S, v \in V \setminus S\}\right|$. Line graph $L(G)$ of graph G is a graph such that each vertex of $L(G)$ represents edge of G ; and any two vertices of $L(G)$ are adjacent if and only if their corresponding edges are incident, meaning they share a common end vertex, in G .

A canonical definition of a graph cluster does not exist, but it is commonly agreed that a cluster should be a connected subgraph induced by a vertex set S with many internal edges $E(S)$ and few edges to outside vertices in $V \setminus S$ (Šíma & Schaeffer, 2006). Although the meaning of a graph cluster is not exactly defined, measures for comparing the quality of different clustering approaches were developed, see section 8.2.7. Graph clusters are referenced by some authors as communities (Flake et al., 2002; Newman & Girvan, 2003; Newman, 2003b; Palla et al., 2005; Radicchi et al., 2004).

The data-set which is intended to be clustered is often very large and the applied clustering approaches often do not scale well. There are several ways in which a data-set can be large:

- there can be a large number of instances in the data-set
- the instances can have many attributes
- the number of clusters can be very high

Depending on the domain the requirements on the produced clusters can differ. We distinguish the following types of clusters:

- *exact clusters* – the instances belong to exactly one cluster
- *overlapping clusters* – the instances can belong to at least to one cluster
- *probabilistic membership in clusters* – the instances are members of clusters with some probability
- *hierarchical clusters* – the identified clusters are further clustered and thus a hierarchical structure of clusters is generated

Figure 8-5 depicts types of clusters.

The problem of clustering can be defined as a problem of finding minimal cut in the graph (Flake et al., 2002). However, this approach is not considered to result in good clusters, therefore additional conditions such as restricting the number of clusters or the size of clusters are necessary. But that changes the original *mincut* problem to NP-hard (Šíma & Schaeffer, 2006).

Clustering algorithms are generally unsupervised learning algorithms. Several communities and thus approaches, such as data mining (k-means, expectation maximization), graph-theoretical (spectral bisection), social network analysis greedy agglomerative and divisive algorithms), contributed to the field. We give an overview of different clustering algorithms.

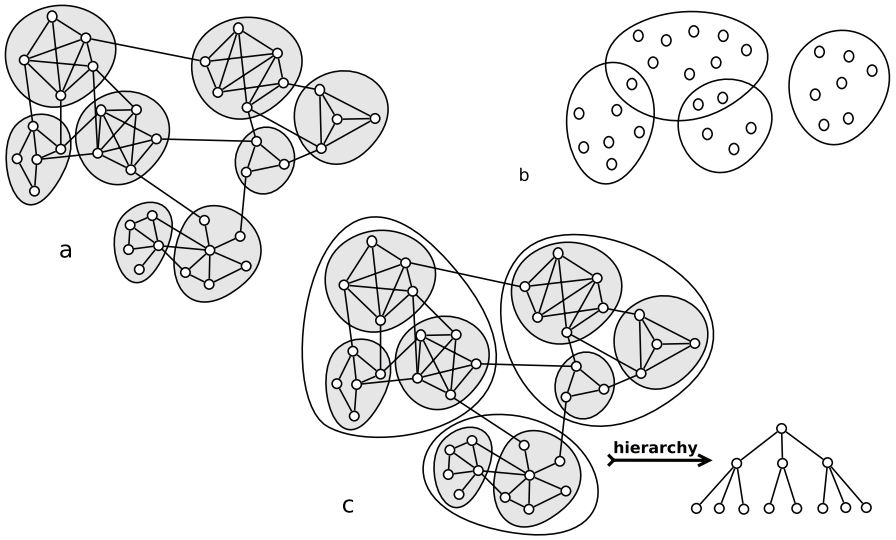


Figure 8-5. Cluster types: a) exact graph cluster; b) overlapping clusters; c) hierarchical clusters.

The following clustering approaches described:

- *Greedy agglomerative or bottom-up approaches* – Vertices are measured for similarity based on different metrics. The most similar vertices are connected to clusters. Therefore the clusters are constructed bottom-up. The most similar vertices are aggregated into a cluster.
- *Greedy divisive or top-down approaches* – Edges with high betweenness (resp. with other property) are removed, the disconnected components are processed further or they are identified as clusters based on a given criterion.
- *Local clustering* – Method also called incremental clustering motivates the use of a local approach for finding a good cluster containing a specified seed vertex or a set of vertices by examining only a limited number of vertices at a time, proceeding in the "vicinity" of the seed vertex.

We restrict to undirected graphs with no self-loops. Some of the described algorithms can be applied on unweighted graphs only. We state the exact type of graph presumed by the algorithms. Estimation of the similarity of the instances and preprocessing of edges between instances (whether they are bridging two clusters or not) is described bellow. We continue with the clustering techniques and latter we conclude with clustering measures useful for clustering quality evaluation and effectiveness.

8.2.1 Instance similarity

Finding the similarity between two instances in a data-set or specifically vertices in a graph is a crucial issue in clustering. For the clustering algorithms it is demanded that similar instances will be placed in the same cluster. For general data, it is not always immediately clear what would be the proper similarity measure. The case is equally confusing on graphs. Computation of similarities possesses often high computational complexity and it is thus not feasible for data-sets with a lot of instances.

Because of the computation of instances' similarity from their position in the network, comparison of attributes is also applied. Instances with similar attribute values are claimed to be more similar. Publication citation clustering or reference matching are generally solved by string similarity measures (McCallum et al., 2000; Aleman-Meza et al., 2006). Reliable string comparison, however, is a costly operation. For proper identification of string's subparts *hidden Markov models* are applied. For comparison of two strings generally *edit distance measure* is used. To tackle the effectiveness related to the expensive similarity measures different optimizing techniques were developed, see section 5.6. We list few ways of measuring the similarity, or rather likelihood of being in the same cluster, of two vertices.

Number of node-independent paths. Two paths are node-independent if they do not contain any common vertex except of the first and the last vertex of the paths. The number of node-independent paths between u and v is equal to the number vertices to be removed to disconnect the two vertices. The higher the number of node-independent paths is, the more likely it is that u and v is in the same cluster.

Total number of paths. In contrast to the previous approach, here the total number of paths (which may contain the same vertices as well) is taken to account. The total number of paths can be infinite (if the graph contains circle between the measured vertices). Therefore the paths should be weighted, so that the final value of weights should attain convergence.

$$W = \sum_{l=0}^{\infty} (\alpha E)^l = [I - \alpha E]^{-1}$$

In the above equation α has to be smaller than the reciprocal of the largest eigenvalue of E (Girvan & Newman, 2001).

Euclidean distance.

$$W[i,j] = \sqrt{\sum_{k \neq i,j} (E[i,k] - E[i,j])^2}$$

The Euclidean distance measurement is a dissimilarity measurement. If two vertices share the same neighborhood then the value of the above equation for them is zero. If more differences arise, the value becomes higher. Algorithms which process the vertices based on their similarity (like GAC in section 8.2.3), matrix $-W$ should be used instead.

Pearson correlation between columns of adjacency matrix. This similarity measurement assumes that the context, the neighbors of vertices belonging to the same cluster are similar. Therefore their columns (or rows) correlate. Vertices showing high correlation in their structural context are supposedly more likely to be in the same cluster.

$$\begin{aligned} \mu_i &= \frac{1}{n} \sum_j E[i,j] \\ \sigma_i^2 &= \frac{1}{n} \sum_j (E[i,j] - \mu_i)^2 \\ W[i,j] &= \frac{\frac{1}{n} \sum_k (E(i,k) - \mu_i)(E(j,k) - \mu_j)}{\sigma_i \sigma_j} \end{aligned}$$

8.2.2 Centrality and identification of bridging edges

There are various measures of the centrality of a vertex within a graph that determine the relative importance of a vertex within the graph. Several centrality measures exist (Brandes, 2001). The measures are defined for vertex measurement, but they can be simply defined also for edges and the computation deduced from vertex computation.

Edge centrality is used by clustering algorithms which rely on identification of edges which are bridging clusters. We also introduce a simple centrality, based on a micro-cosmos model.

Betweenness centrality

Among the variety of different centrality measures betweenness centrality (referenced as shortest-path betweenness by some authors) is the most acclaimed measure used in edge-removal clustering. We use $d_G(u,v)$ to denote the distance between vertices u and v, i.e. the minimum length of any path connecting s and t in G. Let $\sigma_{u,v} = \sigma_{v,u}$ denote the number of shortest paths from $u \in V$ to $v \in V$, where $\sigma_{u,u} = 1$ by convention; $\sigma_{u,v}(w)$ denotes the number of shortest paths from u to v that some $w \in V$ lies on.

$$C_S(v) = \sum_{s \neq v \neq t \in V} \frac{\sigma_{st}(v)}{\sigma_{st}}$$

Brandes describes a fast algorithm based on breadth-first search for computation of centrality values (Brandes, 2001). The algorithm was independently published also by Newman. The spatial complexity of the algorithm is $O(n+m)$, its time complexity is $O(nm)$ on unweighted graphs. More illustrative explanation of the algorithm and betweenness centrality measures is given in (Newman, 2004). Betweenness centrality is known to be strongly correlated with vertex degree in most networks questioning the usefulness of this time-expensive ranking (Newman, 2003a).

The above definition of betweenness centrality is given for vertices, being coherent with other literature. However, we needed the edge betweenness centrality for our clustering purposes. The edge-betweenness is gained by computing the betweenness of the line-graph of the graph.

An interesting alternative to betweenness centrality is described in (Newman, 2003a). The approach is based on random walks and the intuition behind is given by spreading of information in networks. Edges bridging two clusters are supposed to be visited more often than intra cluster edges therefore should get higher ranks.

Using distances after layouting as ranks

We took advantage of Fruchterman-Reingold layouting algorithm (Fruchterman & Reingold, 1991). The algorithm is inspired by natural systems such as springs and macro-cosmic gravity.

Vertices of graph are modeled as atoms, exerting attractive and repulsive forces on each other. The forces between vertices were chosen with the aim to place vertices connected by an edge closer, while vertices not connected by an edge farther from each

other. In this model only adjacent vertices attract each other, while all vertices repel each other. The attractive and repulsive forces are defined as follows:

$$F_{attr}(u,v) = \begin{cases} \frac{\|p_u - p_v\|^2}{k} & if\ (u,v) \in E \\ 0 & otherwise \end{cases}$$

$$F_{rep}(u,v) = \frac{k^2}{\|p_u - p_v\|}$$

where p_u is the position vector of vertex u and u is a constant defined as $k = C\sqrt{\frac{area}{|G(V)|}}$, where C is a constant found experimentally.

The algorithm iterates over the vertices of graph for a defined number of loops. For each vertex it computes the sum of attractive forces from adjacent vertices and the sum of repulsive forces from all vertices. Then the displacement is computed as the sum of attractive and repulsive forces limited by temperature according to cooling schedule.

The grid variant is a modification to the original algorithm in such manner that only vertices within some specified distance repel each other. To avoid examining all vertices on distance from current vertex a so called "grid" data structure (multi-dimensional array) keeps info about vertices in the neighborhood. With a little more memory usage, only vertices in neighboring grid cells have to be examined on distance and the repulsive force is computed only from those that fall within the distance. This makes the algorithm much more computationally effective and suitable also for disconnected graphs. The Euclidean lengths, or our "micro-cosmos centrality", of edges are calculated from the positions of vertices.

8.2.3 Greedy agglomerative clustering

Greedy agglomerative clustering (GAC) is a bottom-up method. The idea is to add edges connecting similar vertices. The algorithm starts with a graph without edges. By addition an edge two components become connected. The components of the graph represent clusters. As the clusters are built from bottom to up, finally the whole graph unites in one component. During the computation a tree structure is produced, which we call *dentogram*, see Figure 8-7.

Algorithm on Figure 8-6 returns a dentogram of clusters which is cut at a given depth of the tree (the dashed line on Figure 8-7). The depth is a tunable parameter of the algorithm. The higher the depth is set, the elaborated the clusters become.

By application of GAC vertices connected by a single edge, placed on the periphery of the cluster or on the other hand vertices in the centre of the cluster are often misclassified (Girvan & Newman, 2001). That makes this approach not suitable for every application. Choosing the right similarity measurement is crucial for the given domain. If we consider that the similarity measure takes $O(1)$ time, computing $O(n^2)$ similarities between all vertex pair needs the same time. Sorting these values therefore needs $O(n^2 \log n)$ time. The dentogram can be created in near linear time

using the tree-based union/find algorithm. Thus the algorithm scales as $O(n^2 \log n)$. GAC does not need to know any idea about the number of the clusters. On the other hand it does not give us any idea, at which depth it is optimal to cut the dentogram.

```
Data: graph G(V,E)
Result: hierarchical clustering - dentogram D
1 Initialise G' = {G(V), ∅};
2 compute matrix W of vertex similarity;
3 while the graph has more than one component do
4   choose vertices u , v where W[u,v] = max_{(i,j)∉G'(E)} W[i,j];
5   add edge (u,v) to G'(E);
6 if edge (u,v) connected two components C_1 and C_2 then
7   add to dentogram D component/cluster C_1 ∪ C_2 with two child
        components C_1 and C_2;
```

Figure 8-6. Hierarchical clustering algorithm.

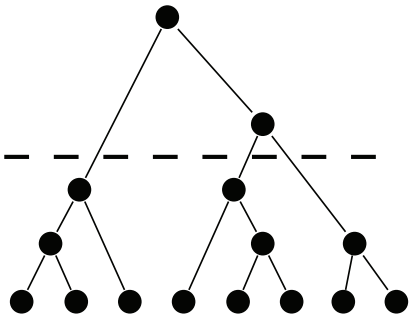


Figure 8-7. Dentogram. The dashed line cuts the cluster tree, the topmost clusters under the line are presented as result.

8.2.4 Greedy divisive clustering

Greedy divisive clustering (GDC), also known as edge removal clustering, first preprocess the edges, and rank them. Based on the ranks the edges are removed from the graph. The emerging components isolated from the rest of the graph are claimed to be clusters. *Newman-Girvan clustering* use edge-betweenness centrality as edge ranks. The general GDC's metacode is depicted in algorithm on Figure 8-8.

```
Data: graph G(V,E)
Result: clustering C = (C_1, C_2, ...)
1 rank edges in E;
2 while the stopping criterion is not fulfilled do
3   remove the edge with the highest rank from E;
4   find all isolated components in G;
5   declare components clusters: C = components(G);
6 return C;
```

Figure 8-8. Edge-removal clustering algorithm.

The *stopping criterion* can be a simple one, which terminates the algorithm, when the expected number of clusters emerged. Huberman et al. (Huberman et al., 2003) proposed a procedure, which stops division of a cluster if it is smaller than 6 vertices or the highest betweenness of any edge is equal to or less than $N-1$, where N is the cluster's size.

8.2.5 A simple local clustering approach

Clustering algorithms based on local search compute clusters from a small set of seed vertices. Several advantages support local clustering approaches:

- *sublinear complexity* – not all of the vertices have to be visited during the clustering,
- *high scalability* – huge, even not explorable networks can be clustered, as only the vicinity of the vertex seed of interest is explored,
- *incremental processing* – the clustering can be suspended at any time yielding the best found cluster.

In spite of these advantages local clustering has also drawbacks. Local clustering is generally less accurate than global clustering methods as it tends to find a local minimum. For tackling this problem several methods were suggested (Schaeffer, 2006). We describe a simple local clustering approach, which is based on an algorithm developed from breadth first search, in which the items in queue have their priorities changing during the processing (Frivolt & Bieliková, 2005). The priority of a vertex is equal to the proportion of the number of neighbors already assigned to the cluster and the total number of adjacent vertices. The metacode of the algorithm is depicted on Figure 8-9.

```
Data: graph G(V,E), seed of initial vertices S, maximum number
of vertices to cut L
Result: set of cut-out vertices C
1  C = S;
2  list = S;
3  while list is not empty and |C| < L do
4     v = pop(list);
5     add v to C;
6     forall neighbors k of v do
7       if k ∉ C then
8         if k ∉ list then
9           rank k in list with priority: |N_k ∩ C| / |N_k|;
10        else
11          add k to list with priority: 1 / |N_k|;
12 return C;
```

Figure 8-9. A simple local clustering algorithm based on breadth first search.

8.2.6 Effectiveness

As it was shown in overview of several clustering techniques in the previous sections we might often face scalability problems during clustering. We recall that the clustered data-set scale on three axes: 1) large number of instances, 2) many or too complex instance attributes, 3) lot of clusters. Local clustering aims to tackle scalability problems related to high number of instances.

The second mentioned scalability problem occurs if estimation of the proper similarity among the instances is a too expensive operation (string comparison). Reliable identification of similarity between scientific citations is a very expensive operation, although the dissimilarity of two citations can be computed very cheaply. A two-pass algorithm was developed for application when dissimilarity can be estimated much more effectively than similarity between a vertex pair (McCallum et al., 2000). The method is generic enough to be well combinable by other approaches. The idea is to create so called *canopies* in the first pass of the algorithm. Canopies are a kind of rough overlapping clusters which can be computed very effectively even if we compare all n^2 pairs. In the second pass, a rigorous, but more expensive, similarity function is executed on every instance pair, which occurs in the same canopy (we remind, that an instance can be included in more than one canopy). Virtually any instance-based clustering algorithm can be used in the second pass of the algorithm. From a theoretical standpoint, by setting certain properties of the inexpensive distance metric, accurate clustering solution can still be recovered with the canopies approach (McCallum et al., 2000). The necessary guarantees needed for avoiding the decay of the clustering quality depend from the clustering algorithm chosen in the second pass.

If c is the number of canopies, f is the average number of canopies to which the instances are included, k is the number of clusters and n the number of instances, the complexity of the canopies approach combined with GAC is $O\left(f^2 n^2 / c\right)$ and $O\left(nkf^2 / c\right)$ with k-means or expectation-maximization.

8.2.7 Clustering measures

The clustering measures define fitness functions f_G defined on the set of possible solutions of the clustering algorithm $f_G(C) \in \Re_o^+$.

Modularity fitness function $Q(C)$ for particular clustering $C = (C_1, C_2, \ldots)$ is defined as follows. Let $e_{i,j}$ be the fraction of edges in the network that connect vertices in cluster C_i to those in cluster C_j, and let $a_i = \sum_j e_{i,j}$. Then

$$Q(C) = \sum_i \left(e_{i,i} - a_i^2\right)$$

is the fraction of edges that fall within clusters, minus the expected value of the same quantity if edges fall at random without regard for the clustering C.

The prominent position among graph cluster measures is occupied by the *conductance* (Šíma & Schaeffer, 2006; Brandes et al., 2003) which is defined for any cut; $0 \neq S \subseteq V$ in graph G as follows:

$$\phi(S) = \frac{c_G(S)}{\min(\deg_G(S), \deg_{G \setminus V}(S))}$$

Based on the notion of conductance, we can now define *inter-cluster* $\delta(C)$ and *intra-cluster conductance* $\alpha(C)$.

$$\begin{aligned} \alpha(C) &= \min_{i \in \{i,\ldots k\}} \phi(G[C_i]) \\ \delta(C) &= 1 - \max_{i \in \{i,\ldots k\}} \phi(C_i) \end{aligned}$$

Intra and inter-cluster conductance are two indices founded on the concepts of bottlenecks, i.e., a cluster should not have a sparse cut separating non-trivial parts and in contrast, the connection of a cluster to the remaining graph should be sparse. Conductance was introduced by random walk theory.

References

ALBERT, R – JEONG, H. – BARABÁSI, A.-L. (1999). Diameter of the World-Wide Web. In: *Nature*, Vol. 401, pp. 130-131.

ALEMAN-MEZA, B. – NAGARAJAN, M. ET AL. (2006). Semantic analytics on social networks: experiences in addressing the problem of conflict of interest detection. In: *WWW '06: Proc. of the 15th Int. Conf. on World Wide Web*, ACM Press, pp. 407-416.

BIANCONI, G. – BARABÁSI, A.-L. (2001). Bose-Einstein Condensation in Complex Networks. In: *Physical Review Letters.*

BRANDES, U. (2001). A Faster Algorithm for Betweenness Centrality, In: *Journal of Mathematical Sociology*, Vol. 25, No. 2, pp. 163-177.

BRANDES, U. – GAERTLER, M. – WAGNER, D. (2003). Experiments on Graph Clustering, In: *Proc. of the 11th Annual European Symposium on Algorithms*, Springer-Verlag, Lecture Notes in Computer Science, Vol. 2832, pp. 568-579.

BRODER, A. – KUMAR, R. – MAGHOUL, F. ET AL. (2000). Graph structure in the web. In: *Computer Networks*, Vol. 33, pp. 309-320.

DILL, S. – KUMAR, R. – MCCURLEY, K. S. ET AL (2002). Self-Similarity in the Web. In: *ACM Transactions on Internet Technology*, Vol. 2, pp. 205-223.

ERDŐS, P. – RÉNYI, A. (1960). Random Graphs. In: *Publ. Math. Inst. Hung. Acad. Sci.*, Vol. 4.

FLAKE, G. – LAWRENCE, S. – GILES, C. L. ET AL (2002). Self-Organization of the Web and Identification of Communities, In: *IEEE Computer*, Vol. 35, No. 35, pp. 66-71.

FRIVOLT, GY. – BIELIKOVÁ, M. (2005). A community-cutting approach, In: *RAWS 2005 – Proc. of the 1st Int. Workshop on Representation and Analysis of Web Space*, pp. 49-54.

FRUCHTERMAN, T. M. J. – REINGOLD, E. M. (1991). Graph Drawing by Force-directed Placement, In: *Software - Practice and Experience*, Vol. 21, No. 11, pp. 1129-1164.

GIRVAN, M. – NEWMAN, M. E. J. (2001). Community structure in social and biological networks, http://arxiv.org/abs/cond-mat/0112110.

HUBERMAN, B. A. – TYLER, J. – WILKINSON, D. (2003). Email as Spectroscopy: Automated Discovery of Community Structure within Organizations, In: *Communities and Technologies*, http://www.citebase.org/cgi-bin/citations?id=oai:arXiv.org:cond-mat/0303264.

MCCALLUM, A. – NIGAM, K. – UNGAR, L. H. (2000). Efficient clustering of high-dimensional data sets with application to reference matching. In: *KDD '00: Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, ACM Press.

NEWMAN, M. E. J. (2000). Models of Small World (a review). In: *Physical Review Letters*, cond-mat/0001118.

NEWMAN, M. E. J. (2003a). A measure of betweenness centrality based on random walks, http://www.citebase.org/cgi-bin/citations?id=oai:arXiv.org:cond-mat/0309045.

NEWMAN, M. E. J. (2003b). Fast algorithm for detecting community structure in networks. http://arxiv.org/abs/cond-mat/0309508.

NEWMAN, M. E. J. (2003c). The structure and function of complex networks. In: *Physical Review Letters*, cond-mat/0303516.

NEWMAN, M. E. J. – GIRVAN, M. (2003). Finding and evaluating community structure in networks. In: *Physical Review E*, Vol. 026113, No. 69.

NEWMAN, M. E. J. (2004). Detecting community structure in networks, In: *The European Physical Journal B*, Vol. 38, pp. 321-330.

PALLA, G. – DERENYI, I. – FARKAS, I. ET AL. (2005). Uncovering the overlapping community structure of complex networks in nature and society, In: *Nature*, Vol. 435, pp. 814.

RADICCHI, F. – CASTELLANO, C. – CECCONI, F. ET AL. (2004). Defining and identifying communities in networks, In: *PROC.NATL.ACAD.SCI.USA*, Vol. 101, pp. 2658.

SCHAEFFER, S. E. (2006). Algorithms for nonuniform networks, *PhD thesis*, Helsinki University of Technology, Department of Computer Science and Engineering.

ŠÍMA, J. – SCHAEFFER, S. E. (2006). On the NP-Completeness of Some Graph Cluster Measures. In: *SOFSEM 2006: Theory and Practice of Computer Science*.

WATTS, D. J. – STROGATZ, S. H. (1998). Collective dynamics of small-world networks. In: *Nature*, Vol. 393, pp. 440-442.

WATTS, J. D. (2003). Small Worlds: The Dynamics of Networks between Order and Randomness, Princeton University Press.

9 ZNOVUPOUŽITIE NÁVRHOVÝCH VZOROV NA ÚROVNI ZDROJOVÉHO KÓDU

Jednou z najdôležitejších súčastí v dnešnom softvérovom inžinierstve je nepochybne znovupoužitie. Znovupoužitie v rôznych kontextoch, rôznych fázach i na rôznych úrovniach abstrakcie.

Pre podporu znovupoužitia vznikajú nové jazyky, nové knižnice, nové katalógy, nové postupy. Všetko smeruje k podpore znovupoužitia a tým k úzko súvisiacemu skracovaniu doby vývoja výsledného produktu - či už hovoríme o dokumente analýzy, o dokumente návrhu, samotnej implementácii produktu, jeho dokumentácii alebo o dokumente výsledkov testov.

Či už hovoríme o šablónach dokumentácie, o metodikách podporujúcich rôzne fázy vývoja softvéru alebo o samotných knižniciach či frameworkoch, stále ide o znovupoužitie zdrojového kódu. Znovupoužitie ale nie je doménou výlučne softvérového inžinierstva a jeho paralely môžeme nájsť i v rôznych odvetviach. Nie sme jediní, ktorí opakujú postupy, zaznamenávajú a katalogizujú myšlienky. Jasným dôkazom je určite i vznik návrhových vzorov.

Alexander ako prvý opísal a katalogizoval vzory, no nebolo to v kontexte softvérového inžinierstva ale v architektúre a územnom plánovaní napr. v (Alexander et al., 1977, Alexander et al., 1979). Na rozdiel od softvérového inžinierstva, kde sa jeho publikácie stali inšpiráciou pre vznik rôznych katalógov, v oblasti architektúry bol odmietnutý.

Definícia vzoru ako takého sa mierne líši od publikácie k publikácii, no podstata znovupoužitia riešenia problému v definovanom kontexte zostáva zachovaná. V oblasti návrhu najlepšie na Alexandra naviazal Gamma a už vo svojej dizertačnej práci a neskôr v publikácii (Gamma et al., 1995) definoval 23 návrhových vzorov, pričom jasne charakterizoval i akýsi vzorový postup ich použitia.

Návrhové vzory sú v súčasnosti implicitne spájané s objektovo-orientovaným vývojom, konkrétne s fázou podrobného návrhu. V (Gamma et al., 1995), kde sa návrhové vzory opisujú semiformálnym spôsobom, sa pre ich zápis a charakteristiku používa najmä diagram tried a väzby ako dedičnosť, asociácia a agregácia, čo jasne vyjadruje zviazanie s objektovo-orientovaným programovaním. Do fázy návrhu návrhové vzory vstupujú ako skupina štandardných tried a vzťahov medzi nimi.

Ďalej sa táto štúdia zaoberá možnosťami znovupoužitia návrhových vzorov, resp. ich častí, pričom upozorňuje na vzory v rôznych oblastiach od CASE nástrojov až k rozšíreniam implementačných jazykov. Analyzujeme znovupoužitie návrhových vzorov s pomocou CASE nástrojov, prostriedkov štandardných a neštandardných objektovo-orientovaných jazykov ako i špeciálnych rozšírení objektovo-orientovaných jazykov.

Opierame sa o viacero publikácií z oblasti návrhových vzorov, objektovo-orientovaného vývoja a znovupoužitia. Vychádzame z publikácií prezentovaných na rôznych lokálnych i medzinárodných konferenciách.

9.1 Znovupoužitie vzorov

9.1.1 Znovupoužitie vzorov na úrovni myšlienok

Vzory ako také sa v súčasnosti definujú najčastejšie semiformálne, aj keď existujú isté iniciatívy o definíciu vzorov na formálnej úrovni, napr. (Dietrich & Elgar, 2005, Eden, 2002, Kim et al., 2003, Taibi & Ceck Ling Ngo, 2003). Vzory sa katalogizujú do rôznych zbierok či už v elektronickej podobe napr. (Cunningham, 2006, D'Adderio et al., 2006) alebo v podobe rôznych publikácií ako napr. (Al-Ahmad, 1997, Alur et al., 2001, Bushman et al., 1996, Gamma et al., 1995, Gil & Maman, 2005, MacDonald et al., 2002). Už z viacerých definícií samotného vzoru je zrejmé, že vzory sa používajú na úrovni zachytených riešení, myšlienok pri riešení, pričom pri použití vzoru sa nedostávame do problému stereotypného opakovania riešenia. Môžeme povedať, že inštancia vzoru (nezávisle od toho, o aký typ vzoru ide – či už analytický, architektonický, implementačný alebo iný), sa počas vytvárania riešenia konkrétneho problému ohýba podľa aktuálnych obmedzení známych v kontexte navrhnutého riešenia. Práve takáto aplikácia vzoru definuje akýsi nedogmatický prístup k použitiu návrhových vzorov.

Vzor sa teda použije pri výskyte konkrétneho problému v konkrétnom kontexte, ktorý sa podobá problému opísanému v katalógu. Mierne modifikácie skryté pod pojmom ohýbanie vzoru sú akceptovateľné a nemenia celkový pohľad na riešenie problému. Takéto znovupoužitie riešenia zapísaného semiformálnym spôsobom vo forme vzoru je výhodne v jednoduchosti znovupoužitia, pričom treba poznať doménovú problematiku ako i samotný vzor, ktorý používateľ ohýba podľa potrieb riešenia.

Nevýhodou môže byť nepovolené ohýbanie vzoru a vznik tzv. mutovanej inštancie vzoru, ktorá mení hlavnú myšlienku riešenia a zavádza používateľa pri pomenovaní elementov riešenia. Ide o inštanciu vzoru, ktorá je na prvý pohľad skutočnou inštanciou konkrétneho vzoru, no vzor ohýba až do tej miery, kedy mení jeho charakteristické črty. Vznik takejto inštancie nemusí byť úmyselný a okamžite po vzniku nevyvolá takmer žiadne problémy. Komplikácie nastávajú až v prípade údržby, po zabudnutí dôvodov ohýbania. Pri použití nedogmatického prístupu nevzniká veľa podobných vzorov, resp. podobné vzory sa považujú za inštancie všeobecného vzoru, čo sa môže považovať za výhodu v zmysle nepotrebnosti študovať a dokumentovať podobné vzory, líšiace sa len v detailoch.

Druhým spôsobom ako pristupovať k teórii vzorov je dogmatický prístup. Od nedogmatického prístupu sa odlišuje v základnom princípe, kedy existujú katalogizované vzory, ktoré sú takmer nemenné a používajú sa iba v presne defino-

vaných problémoch a kontextoch. Používateľ po identifikácii konkrétneho problému a kontextu, v ktorom sa problém vyskytuje, hľadá vzor, ktorý čo možno v najväčšej miere rieši definovaný problém. Po nájdení vzoru sa vzor len v minimálnej miere zmení, ale ústredná časť vzoru zostáva nemenná. Výhodou je rýchlejšie použitie vzoru bez potreby ďalších ohýbaní. Pri používaní dogmatického prístupu nevzniká problém s mutovanou inštanciou vzoru a zavádzaním používateľa definovaným slovníkom, keďže vzory je možné meniť naozaj iba v minimálnej miere. Na druhej strane nevýhodou je potreba štúdia a dokumentovania ďaleko väčšej množiny vzorov ako v prvom prípade, ktoré sa od seba odlišujú iba v drobnostiach.

V tabuľke 9-1 je uvedené porovnanie dogmatického a nedogmatického prístupu k používaniu vzorov na všeobecnej úrovni.

Tabuľka 9-1. Porovnanie dogmatického a nedogmatického prístupu k vzorom.

	počet vzorov	použitie vzoru	vznik mutovanej inštancie vzoru
dogmatický prístup	veľký počet podobných vzorov, nutnosť ich štúdia a dokumentácie	po nájdení vzoru už len minimálna modifikácia a vzor je relatívne rýchlo pripravený na použitie	mutované vzory vznikajú len veľmi málo a náročne, keďže vzor nie je možné ľubovoľne modifikovať
nedogmatický prístup	celkovo menší počet vzorov, podobné vzory vznikajú inštan-covaním a ohý-baním všeobecných vzorov, nutnosť poznať všeobecné vzory na podrobnej úrovni	nutnosť štúdia vzoru, problematiky, po nájdení vzoru je vzor ohýbaný podľa potrieb konkrétnej domény a potom pripravený na použitie	vzor môže byť modifikovaný v akomkoľvek smere, v prípade nesprávnej modifikácie môže dôjsť k vzniku tzv. mutovaného vzoru

Z teoretického pohľadu môžeme hovoriť o dvoch hlavných možnostiach prístupu k použitiu vzorov. Problém výberu konkrétneho prístupu je ale tiež ovplyvnený i množinou vzorov, pre ten ktorý prístup v tej ktorej fáze vývoja softvéru. Všeobecne sa dá povedať, že nedogmatický prístup je v zásade použiteľný v ľubovoľnej fáze a úrovni abstrakcie použitia vzorov. Dogmatický prístup vyžaduje k svojmu použitiu väčšiu množinu vzorov ako i formálnejšiu definíciu samotných vzorov.

V zásade budeme v ďalej hovoriť o nedogmatickom prístupe k vzorom, pričom hlavnú snahu sústredíme na kombináciu týchto dvoch prístupov. Umožniť modifikovať to, čo sa líši od inštancie k inštancii a umožniť znovupoužitie všeobecných častí vzorov.

9.1.2 Znovupoužitie návrhových vzorov na úrovni myšlienok

Návrhové vzory sa v ničom nevymykajú všeobecnej definícii vzoru. Používajú sa vo fáze návrhu softvérových produktov, pričom v niektorých prípadoch by sme k nim mohli zaradiť i architektonické vzory. My sa budeme ďalej zaoberať výlučne

návrhovými vzormi zodpovedajúcimi detailnému návrhu v kontexte objektovo orientovaného vývoja softvéru a budeme vychádzať najmä z katalógu (Gamma et al., 1995).

V zásade neexistujú žiadne obmedzenia, ktoré by uprednostnili dogmatický alebo nedogmatický prístup k návrhovým vzorom. Otázka môže znieť: ako a kedy sa aplikujú návrhové vzory a ktorý z prístupov v tom alebo onom prípade použiť? Opäť sú zaužívané dva základné prístupy kedy a ako návrhové vzory použiť. Pomôckou alebo akousi metodikou v prípade použitia návrhového vzoru môžu byť rôzne metódy pre prácu so vzormi napr.: (Dong, 2002, Jakubík, 2005a, Yacoub & Ammar, 2000), ktoré sú viac alebo menej zdokumentované v rôznych publikáciách.

Návrhové vzory sa môžu použiť počas štandardnej fázy detailného návrhu, kedy sa podľa požiadaviek a predchádzajúcej analýzy vzory vyberajú na riešenie konkrétnych problémov. Uprednostnenie použitia dogmatického alebo nedogmatického prístupu úzko závisí od konkrétneho prípadu. Ak navrhujeme softvér takpovediac na zelenej lúke, zdá sa vhodnejšie použiť dogmatický prístup, počas ktorého sú jasne identifikované vzory, jednotlivé roly vo vzoroch ako i funkcionalita spojená s konkrétnym vzorom. V prípade využitia už vytvorených častí softvéru, ktoré sa budú meniť pre potreby aplikácie, je asi vhodnejšie použiť nedogmatický prístup, s ktorým zjednodušíme samotný vývoj, umožníme prispôsobiť samotný vzor konkrétnemu použitiu, no vyžadujeme kvalifikovaného návrhára ovládajúceho definovanú skupinu návrhových vzorov.

Ak návrhové vzory používame v časti refaktoringu a optimalizácie (aj keď nie výkonnostnej, keďže súčasné návrhové vzory skôr pridávajú do návrhu ďalšie vrstvy abstrakcie, nové triedy a samotné vykonanie funkcionality nie je optimalizované z hľadiska času a výkonu) sú vzory častokrát modifikované v existujúcom návrhu alebo vkladané do častí návrhu, kde vyžadujeme vyššiu generalicitu a všeobecnosť. Pridávanie genericity a všeobecnosti v konečnom dôsledku spôsobuje ľahšiu rozšíriteľnosť a jednoduchšiu konfigurovateľnosť výsledného softvérového produktu. V tomto prípade sa zdá vhodnejší nedogmatický prístup z dôvodu možných modifikácií všeobecných vzorov, bez potreby prílišného upravovania už existujúcich častí systému.

V tabuľke 9-2 uvádzame vybrané vlastnosti použitia návrhových vzorov v rôznych etapách vývoja softvéru pri rôznych prístupoch.

9.1.3 Znovupoužitie návrhových vzorov na úrovni zdrojového kódu

Predchádzajúce časti sú zamerané na objasnenie všeobecných prístupov a metód použitia či už vzorov alebo konkrétne návrhových vzorov. V nasledujúcich častiach sa budeme zaoberať konkrétne možnosťami znovupoužitia častí návrhových vzorov.

Objektovo orientované programovanie má ako jeden z cieľov umožniť jednoduché a z hľadiska vývoja efektívne znovupoužitie častí systémov. K naplneniu tohto cieľa môžeme v OO programovaní využívať dedenie, polymorfizmus, asociácie a ďalšie viac alebo menej elementárne prostriedky. Návrhové vzory, ktorými sa tu zaoberáme, sa implicitne spájajú s objektovo orientovaným programovaním, no ich znovupoužitie je neformálne definované iba na úrovni postupov, myšlienok a konceptov riešení.

Tabuľka 9-2. Dogmatický a nedogmatický prístup pri použití návrhových vzorov.

	nový návrh	refaktoring existujúceho návrhu
dogmatický prístup	– rýchlejší vývoj vzhľadom na minimálne možnosti modifikácie konkrétnych vzorov – spomalený vývoj vzhľadom na nutnosť naštudovať väčšie množstvo konkrétnych vzorov – v prípade potreby nutné vytvárať a dokumentovať nové vzory	– náročné použitie vzhľadom na existujúci kód, ktorý nemusí vyhovovať žiadnemu z existujúcich vzorov – spomalený vývoj vzhľadom na potrebu zmeny viacerých elementov vstupujúcich do vzoru
nedogmatický prístup	– väčšia voľnosť pri vývoji – potrebné študovať iba všeobecné vzory – nutnosť študovať všeobecné vzory na vyššej úrovni podrobnosti – spomalený vývoj vzhľadom na potreby ohýbania vzorov	– možnosť návrhový vzor ohýbať podľa potrieb existujúceho návrhu – čiastočné zmeny vzoru pri ohýbaní a čiastočné zmeny existujúcich častí systému pre potreby návrhového vzoru

Otázka teda je, či je možné znovupoužiť časti návrhových vzorov na nižšej úrovni? Keďže sa pohybujeme na úrovni podrobného návrhu systému, sme blízko k implementácii, budeme sa zaoberať znovupoužitím častí návrhových vzorov na úrovni zdrojového kódu, ktorá je samozrejme úzko spojená so samotnou etapou podrobného návrhu.

Jedným z cieľov bude identifikovať elementy návrhových vzorov, ktoré sú vo väčšine prípadov doménovo nezávislé a opakujúce sa nezávisle od kontextu. Na dosiahnutie tohto cieľa bude potrebné analyzovať rôzne prostriedky od CASE nástrojov (v časti 9.3), cez špecifické jazyky (v časti 9.4) až k rozšíreniam jazykov (v časti 9.6) po prípade navrhnúť a realizovať nový, efektívnejší a prehľadnejší spôsob, akým sa dá definované rozdelenie dosiahnuť.

V nasledujúcej časti definujeme potrebné pojmy pre rozdelenie návrhového vzoru a vysvetlíme základné problémy spojené s rozdelením.

9.2 Dekompozícia vzoru

Návrhové vzory ale i vzory vo všeobecnosti sa nezaoberajú možnosťami znovupoužitia ich častí. Pri analýze ich štruktúr sa dá povedať, že veľmi úzko spájajú všeobecné, doménovo nezávislé a špeciálne, doménovo závislé časti.

Pod všeobecnou časťou rozumieme tú časť štruktúry návrhového vzoru, ktorá sa opakuje v rôznych kontextoch použitia vzoru a teda zostáva nezmenená v rôznych inštanciách vzoru. Všeobecná časť je súčasťou úložiska vzorov, z ktorého sa

znovupoužíva v rôznych aplikáciách, pričom môže byť znovupoužitá vo viacerých inštanciách vzoru v rámci jednej alebo viacerých aplikácií.

Pod doménovo závislou časťou rozumieme tú časť štruktúry návrhového vzoru, ktorá je špecifická pre konkrétnu doménu a je závislá od použitia vzoru. Doménovo závislá časť vzoru je súčasťou aplikácie a pomocou nej sa návrhový vzor prispôsobuje konkrétnej aplikácii. Rozdelenie na všeobecnú a doménovo závislú časť si ďalej ukážeme na príklade rozdelenia vzoru *Composite* vychádzajúceho z (Jakubík, 2005b), ktorý sme prezentovali na konferencii Objekty 2005.

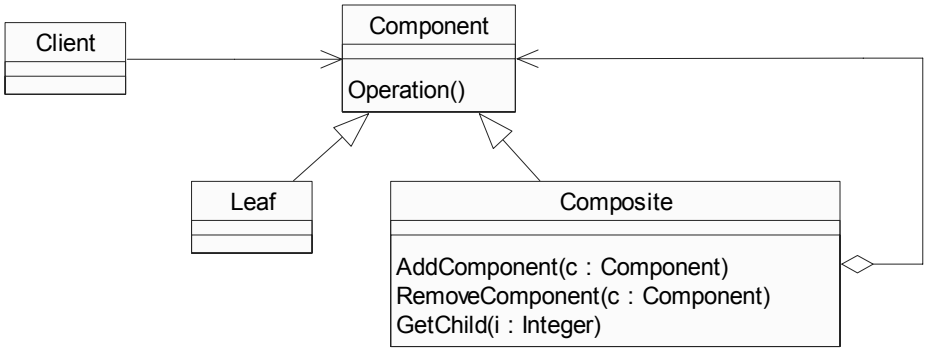
9.2.1 Variácie vzoru Composite

Ako príklad pre rozdelenie vzoru na všeobecnú a doménovo závislú časť použijeme často aplikovaný vzor *Composite* definovaný v (Gamma et al., 1995). *Composite* zabezpečuje skladanie objektov do stromových štruktúr k vyjadreniu hierarchií. *Composite* umožňuje klientom zaobchádzať rovnako s jednotlivými objektmi ako aj s ich zloženinami. (Gamma et al., 1995) Pod zloženinou sa v tomto prípade rozumie skupina elementárnych prvkov a/alebo skupina ich zložení. Ide teda o rekurzívnu štruktúru.

Návrhový vzor *Composite*, v doslovnom preklade Skladba, má dve základné implementácie:

- bezpečná skladba
- transparentná skladba

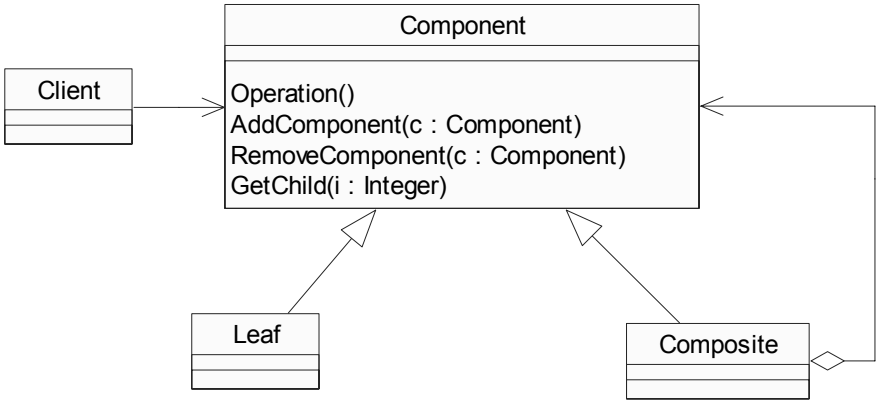
Hlavný a najmarkantnejší rozdiel je v definovaní metód súvisiacich s kontajner manažmentom (riadením nádoby). Pri bezpečnej skladbe uvedenej na obrázku 9-1 sa metódy pre pridávanie, odstraňovanie a sprístupňovanie prvkov kontajnera (nádoby) definujú až na úrovni samotnej triedy *Composite*. Implementácia saoznačuje ako bezpečná, pretože nie je možné volať metódy kontajnera (nádoby) pre elementárne prvky skladby (listy). Na druhej strane sa však porušuje princíp rovnakého prístupu k elementárnym prvkom a k ich zloženinám.



Obrázok 9-1. Bezpečná skladba.

Pri transparentnej skladbe uvedenej na obrázku 9-2 sú metódy pre manažment obsahu kontajnera definované už v triede *Component* a teda je vytvorené prostredie pre rovnaký prístup aj k listom aj k zloženým štruktúram. Nevýhodou v tomto prípade je

nutnosť ošetrenia práce elementárneho prvku ako kontajnera (pridávanie hierarchie listov k inému listu, v ktorom sa táto funkcionalita vôbec nepodporuje).



Obrázok 9-2. Transparentná skladba.

Implementácia tejto variácie vzoru *Composite* sa môže realizovať rôznymi spôsobmi, či už sú problémové metódy ošetrené default implementáciou v triede *Component*, alebo sa ich implementácia ponecháva na triedy nižšie v hierarchii. Keďže ale definícia rozhrania listov a tiež zložení umožňuje volanie metód spojených s kontajner manažmentom, musí sa na niektorej z vrstiev modelu dedičnosti v hierarchii tried nachádzať implementácia týchto metód pre každý z listov pod triedou *Component*.

Obe implementácie majú svoje výhody i nevýhody, ktorými sa na tomto mieste nebudeme zaoberať. Podrobnejšie informácie k obom implementáciám sú k dispozícií napr. v (Geary, 2002) a samozrejme v (Gamma et al., 1995).

V oboch implementáciách je na bližší pohľad zrejmé, že práve metódy spojené s manažmentom obsahu kontajnera sa môžu vo viacerých prípadoch použitia vzoru opakovať. V každej inštancii vzoru potrebujeme kontajner pre ukladanie elementárnych prvkov v zloženinách, no potrebujeme i modifikovať, v závislosti od domény použitia, názvoslovie či už ide o názvy tried, metód alebo atribútov.

9.2.2 Oddelenie všeobecnej a doménovo závislej časti vzoru Composite

Vychádzajúc z vlastných skúseností označíme za doménovo nezávislé časti tried *Component,* resp. *Composite* súvisiace s kontajner manažmentom. Triedu *Leaf* označíme za úplne doménovo závislú.

Ak chceme návrhový vzor použiť s ideológiou výberu z úložiska a pripojeniu doménovo závislých častí, treba osamostatniť všeobecné časti tried, konkrétne v prípade vzoru *Composite* tried *Component* a *Composite*.

Component ale definuje doménové operácie, ktoré nevieme vopred špecifikovať. Operácie definované v *Component* sú implementované i v triede *Composite*.

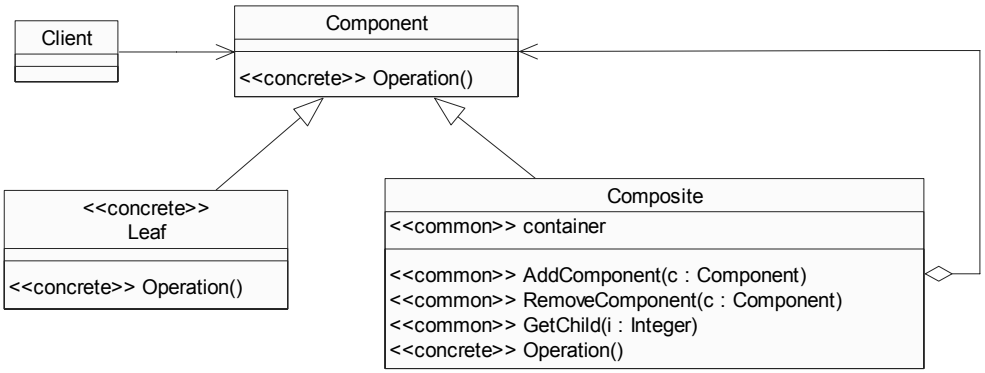
Oddelenie všeobecných častí vzoru pri bezpečnej skladbe

Pri bezpečnej skladbe sa v triede *Component* definujú iba doménovo závislé metódy, ktoré môžu byť čiastočne implementované už na tomto mieste alebo môže byť ich

implementácia prekrytá, resp. presunutá do triedy *Composite* (vykonávanie doménových operácií nad skupinami listov) a súčasne do konkrétnych tried *Leaf* (vykonávanie doménových operácií nad jednotlivými listami stromovej štruktúry). Triedy *Leaf* sú úplne doménovo nezávislé a akási všeobecná časť by sa v nich hľadala len veľmi ťažko, keďže sa menia od inštancie k inštancii vzoru.

Trieda *Composite* obsahuje ďalej operácie spojené s manažmentom obsahu kontajnera ako i samotný kontajner pre uchovávanie skupín tried *Component*, či už listov alebo ich ďalších zložení. Môžeme povedať, že táto časť triedy *Composite* je doménovo nezávislá, pretože pokiaľ nie sme ovplyvnení nejakými ďalšími okolnosťami, napr. výkonnosťou, môžeme použiť ľubovoľný kontajner a naň naviazať základné operácie prístupu k prvkom kontajnera ako pridanie, odstránenie a sprístupnenie Component-ov z kontajnera.

Schéma zvýrazňujúca a graficky oddeľujúca všeobecné a doménovo závislé časti jednotlivých tried bezpečnej variácie vzoru *Composite* je zobrazená na obrázku *9-3*.



Obrázok 9-3. Dekompozícia bezpečnej variácie vzoru Composite.

V diagrame tried na obrázku *9-3* sme pomocou stereotypu <common> označili všeobecnú časť triedy, resp. vzoru a pomocou stereotypu <concrete> označili doménovo závislú časť triedy v kontexte bezpečnej variácie vzoru *Composite*.

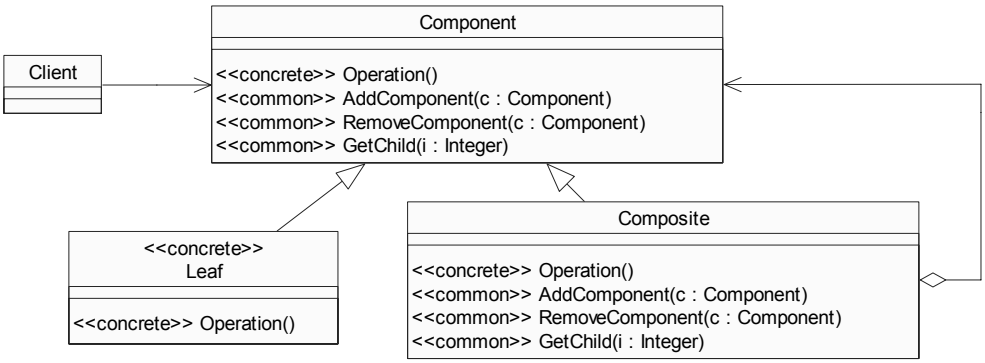
Oddelenie všeobecných častí vzoru pri transparentnej skladbe

Rozdiel medzi bezpečnou a transparentnou skladbou je v umiestnení definícií metód spojených s kontajner manažmentom. V prípade transparentnej skladby sú metódy definované už v triede *Component*. Metódy súvisiace s kontajner manažmentom musia byť implementované v každom liste a tiež v triede *Composite*.

V tomto prípade už trieda *Component* obsahuje všeobecné časti vzoru, ktoré sú v podobe definície metód pre manažment obsahu kontajneru. Od triedy *Component* dedia ďalej triedy *Composite* a jednotlivé *Leaf*. Najmä z hľadiska jednotlivých *Leaf* je vhodné definovať prvotné implementácie metód súvisiacich s manažmentom obsahu kontajnera už v triede *Component*. V triedach *Leaf* potom nemusí byť ošetrovaný problém s metódami kontajner manažmentu. Prvotná implementácia metód môže byť realizovaná na rôznej úrovni od generovania výnimky, cez návrat nulovej hodnoty *NULL* až po prázdne telo metódy.

V triede *Composite* sú implementované podobne ako v predchádzajúcej variácii vzoru i doménovo nezávislé metódy spojené s kontajner manažmentom ako i doménovo závislé metódy definované vo všeobecnej triede *Component*.

Triedy typu *Leaf* sú doménovo nezávislé no musia spĺňať rozhranie definované v triede *Component*.



Obrázok 9-4. Dekompozícia transparentnej variácie vzoru Composite.

Na diagrame tried zobrazenom na obrázku 9-4 sú pomocou stereotypov <common> a <concrete> zvýraznené všeobecné a doménovo závislé časti vzoru *Composite*, konkrétne transparentnej variácie.

9.2.3 Diskusia

Spôsob rozdelenia návrhového vzoru na všeobecnú a doménovo závislú časť si vyžaduje predchádzajúce skúsenosti s používaním návrhového vzoru, pretože je pomerne náročné určiť všeobecnú časť vzoru. Dokonca v niektorých prípadoch špecifických návrhových vzorov nemusí byť dekompozícia vôbec možná.

Problém rozdelenia návrhového vzoru nesúvisí len s predchádzajúcimi skúsenosťami so vzorom, ale i so spôsobom a procesom identifikácie nových návrhových vzorov, ktoré sú častokrát identifikované poloautomaticky v existujúcich návrhoch, odkiaľ prejdú procesom zovšeobecnenia až ku katalogizácii. Celkový prístup k návrhovým vzorom, napriek tomu že sú pomerne blízko k implementácii, neuvažuje o možnostiach ich znovupoužitia na úrovni zdrojového kódu.

Nepodarilo sa nám nájsť žiadny dokument vo forme metodiky alebo postupu, akým návrhový vzor rozdeliť na spomínané časti. Celkovo objem informácií v oblasti znovupoužitia návrhových vzorov na úrovni ich častí je pomerne malý. Veľa publikácií sa sústreďuje na identifikáciu nových vzorov, resp. identifikáciu katalogizovaných vzorov v existujúcich návrhoch.

Pri získavaní informácií z rôznych prameňov sa nám podarilo nájsť jediný dokument (Agerbo & Cornils, 1997) zaoberajúci sa dekompozíciou vzorov podľa (Gamma et al., 1995). Autori opisujú všetkých 23 vzorov podľa (Gamma et al., 1995) a vyjadrujú sa k ich dekompozícii na všeobecnú a doménovo závislú časť s použitím vlastného názvoslovia.

Dá sa teda povedať, že napriek tomu, že vzory sú v softvérovom inžinierstve už niekoľko rokov, iniciatívy v oblasti vytvárania znovupoužiteľných úložísk návrhových

vzorov sú buď nevýrazné alebo zanikli v ich počiatkoch. Ako sa presvedčíme v ďalších častiach, nie je tomu celkom tak.

9.3 Využitie CASE nástrojov pre dekompozíciu vzoru

Pod pojmom dekompozícia vzoru budeme ďalej rozumieť proces rozdelenia návrhového vzoru na všeobecnú a doménovo závislú časť.

Jedným z používateľského hľadiska najjednoduchším spôsobom dekompozície vzoru je využitie možností CASE nástrojov. CASE nástroje podporujú vývoj softvéru v rôznych fázach a rôznym spôsobom. Do tohto kontextu patria CASE nástroje s podporou diagramov tried a návrhových vzorov vo fáze podrobného návrhu. Počas vytvárania modelu v podobe diagramu tried je možné generovať zdrojový kód aplikácie v rôznych implementačných jazykoch (najčastejšie ide o C++, VB, C#, JAVA). Ambíciu pre podporu podrobného návrhu napr. aj vo forme podpory návrhových vzorov má veľké množstvo CASE nástrojov. Pre porovnanie a vyhodnotenie nástrojov sme definovali niekoľko porovnávacích kritérií a podľa nich sme porovnali vybrané nástroje patriace k špičke v súčasnom modelovaní v článku (Jakubík, 2006), ktorý bol prezentovaný na študentskej vedeckej konferencii IIT.SRC 2006 v Bratislave. Zdrojom pre túto časť bol prezentovaný článok (Jakubík, 2006).

9.3.1 Porovnávacie kritériá CASE nástrojov

V nasledujúcich častiach neformálne popíšeme skupinu porovnávacích kritérií pre výber CASE nástroja na základe podpory návrhových vzorov, podpory generovania zdrojového kódu a v neposlednom rade podpory rôznych implementačných jazykov.

Do úvahy nebude brané komerciou jedno z najdôležitejších kritérií, ktorým je cena. Stredobodom záujmu sú návrhové vzory a ich podpora, ktorá môže byť v protiklade s cenou. S narastajúcou cenou častokrát narastá i podpora rôznych špecifických činností v softvérovom inžinierstve. Na druhú stranu sa nedá povedať, že napr. *open-source* projekty nemôžu podporovať prácu s návrhovými vzormi pre ich relatívne nízku obstarávaciu cenu.

Porovnanie a oboznámenie sa s rôznymi CASE nástrojmi pomohlo k pochopeniu a neformálnemu určeniu spôsobu znovupoužitia návrhových vzorov.

Podpora implementačných jazykov

Súčasné CASE nástroje vo väčšine prípadov podporujú implementačné jazyky ako JAVA, C++, C# a VB.

Od CASE nástroja v zásade očakávame generovanie zdrojového kódu v jazyku, resp. V jazykoch, v ktorých sa realizuje celý projekt. Výber nástroja ale často nie je záležitosťou iba jediného projektu a teda pri investícií do kvalitného nástroja budeme očakávať podporu pre čo možno najviac implementačných jazykov.

Podporované návrhové vzory

Medzi najznámejšiu skupinu návrhových vzorov v súčasnosti nepochybne patrí skupina 23 vzorov od Gammu publikovaných v (Gamma et al., 1995).

Vybraný CASE nástroj by mal implicitne podporovať minimálne túto množinu návrhových vzorov, ktoré sú v súčasnosti pokladané za základ v oblasti vzorov. Samozrejme rozšírenie tejto množiny o akýkoľvek ďalší vzor je výhodou.

Generovanie zdrojového kódu

Podpora pre generovanie zdrojového kódu je súčasťou viacerých CASE nástrojov. V zásade existujú dva základné prístupy ku generovaniu zdrojového kódu v konkrétnom implementačnom jazyku:

- zdrojový kód je generovaný po vytvorení modelu na žiadosť používateľa
- zdrojový kód je generovaný v priebehu vytvárania modelu bez účasti používateľa

Oba z prístupov majú svoje výhody i nevýhody. V prvom prípade je potrebné najskôr vytvoriť konzistentný model a potom generovať zdrojový kód, pričom veľakrát je potrebné už na začiatku vytvárania modelu vybrať implementačný jazyk kvôli použitiu správnej skupiny knižníc, rozhraní a ďalších obmedzení daných vybraným jazykom. Výhodou je podpora životného cyklu projektu, kedy je najskôr vytváraný podrobný návrh a k zdrojovému kódu sa dostaneme, až keď sme spokojní s návrhom. Nevýhodou je, že pre aktualizáciu modelu podľa zmeneného návrhového vzoru potrebujeme modul pre spätné inžinierstvo, ktorý nemusí byť súčasťou inštalácie CASE nástroja, resp. môže byť platený samostatne, čo samozrejme zvýši záverečnú cenu CASE nástroja.

V druhom prípade je zdrojový kód generovaný na pozadí počas vytvárania modelu často bez akejkoľvek účasti návrhára. V mnohých prípadoch nie je možné ani len vypnúť toto generovanie zdrojového kódu, čo považujeme za nevýhodu. Výhodou je implicitná podpora spätného inžinierstva, kedy sa zmena v zdrojovom kóde automaticky premietne do modelu.

Existujú aj špeciálne prípady CASE nástrojov ako napríklad vývojový nástroj Visual Studio 2005 od Microsoftu. Modul pre modelovanie diagramov tried nielenže nepodporuje ani základnú notáciu UML, ale aj pracuje priamo so zdrojovým kódom aplikácie a nevytvára žiadny model. V príklade 9-1 je uvedená časť XML súboru, ktorý generuje Visual Studio 2005 ako súbor s diagramom tried. Po analýze jednotlivých častí XML súboru je jasné, že ide iba o záznam informácií o vizuálnej stránke modelu a teda umiestnenie jednotlivých tried v modely.

```
<?xml version="1.0" encoding="utf-8"?>
<ClassDiagram MajorVersion="1" MinorVersion="1">
  <Font Name="Tahoma" Size="8.25" />
  <Class Name="balls.Ball" Collapsed="true">
    <Position X="2.25" Y="1.25" Width="1.5" />
    <TypeIdentifier>
      <FileName>Ball.cs</FileName>
      <HashCode>AEAIAEAAGAAAABCAAAIACgIQAAgBIAAAA=</HashCode>
    </TypeIdentifier>
  </Class>

  <Class Name="balls.Balls" Collapsed="true">
    <Position X="0.5" Y="1.25" Width="1.5" />
    <TypeIdentifier>
      <FileName>Balls.cs</FileName>
      <HashCode>AAAAUQAAASCAAAgAABCAAAACAAAAAAAAA=</HashCode>
    </TypeIdentifier>
  </Class>
```

```
  <Class Name="balls.Program" Collapsed="true">
    <Position X="5.25" Y="0.5" Width="1.5" />
    <TypeIdentifier>
      <FileName>Program.cs</FileName>
      <HashCode>AAAAAAAAAAAAAAAAAAAAAAAAAAAABAAAAAA=</HashCode>
    </TypeIdentifier>
  </Class>
  <Class Name="balls.TestBall" Collapsed="true">
    <Position X="2.25" Y="0.5" Width="1.5" />
    <TypeIdentifier>
      <FileName>TestBall.cs</FileName>
      <HashCode>AAAAAAAACAAAAAACABQASAAAAAAA=</HashCode>
    </TypeIdentifier>
  </Class>
  <Class Name="balls.Properties.Resources" Collapsed="true">
    <Position X="5.25" Y="1.25" Width="1.5" />
    <TypeIdentifier>
      <HashCode>AAAAAAAAAAAAAAAQAAAAAAAAAAAAAAAIA=</HashCode>
    </TypeIdentifier>
  </Class>
  <Class Name="balls.Properties.Settings" Collapsed="true">
    <Position X="5.25" Y="2" Width="1.5" />
    <TypeIdentifier>
      <HashCode>AAAAAAAAAAAAAAAABAAAAAAAAAAAAAAA=</HashCode>
    </TypeIdentifier>
  </Class>
</ClassDiagram>
```

Príklad 9-1. Súbor .cd s informáciami o konkrétnom diagrame tried.

Súbor s koncovkou cd je štandardný XML súbor pričom obsahuje informácie o použitom fonte v popiskoch tried, metód a atribútov (značka Font), informácie o jednotlivých triedach (značka Class) ako meno (atribút Name), pozícia umiestnenia v modeli (značka Position) a jednoznačnú identifikáciu (TypeIdentifier).

Vlastnosti generovaného zdrojového kódu

Toto kritérium patrí zrejme medzi jedno z najdôležitejších, pokiaľ CASE nástroje disponujú rovnakými prostriedkami v zmysle komponentov pre generovanie zdrojového kódu, podpory skupiny návrhových vzorov a ďalšími podobnými, odlišujú sa práve vo vlastnostiach generovaného zdrojového kódu.

Budeme klásť dôraz iba na zdrojový kód návrhových vzorov, v opačnom prípade ide iba o generovanie základných kostier tried (prázdna trieda s metódami podľa modelu). V prípade návrhových vzorov je ale možné generovať i časti metód a práve tu sa prejaví rozdelenie vzoru na všeobecnú a doménovo závislú časť, kedy CASE nástroj má vo svojich vnútorných štruktúrach, ktoré sú z vonku neprístupné, rozdelený vzor na spomínané časti. Pomocou správneho rozdelenia vzoru dochádza k znovupoužitiu preddefinovaných častí návrhových vzorov.

Napríklad pre návrhový vzor *Composite* viaceré CASE nástroje dokázali vygenerovať vrapovanie metód (pridanie prvku, odstránenie prvku, sprístupnenie prvku) triedy *Composite* na *ArrayList*.

Vytváranie inštancie vzoru

Vyzretosť CASE nástroja sa viackrát prejavila v podpore pre vytváranie inštancií návrhových vzorov. Rôzny CASE nástroj rovná sa rôzna podpora pri vytváraní inštancií. CASE nástroje využívajú model rolí použitý napr. v (Lauder & Kent, 1998, Reenskaug, 1995, Riehle, 2000) pre udržiavanie informácií o návrhovom vzore, pričom nad týmito modelmi implementujú sprievodcov, pomocou ktorých sa obsadzujú jednotlivé roly v modeli.

Na jednej strane ide o viackrokových sprievodcov dopĺňaných názorným helpom, ktorý ilustruje vytváranie inštancie a priebežne objasňuje jednotlivé pojmy a roly vo vzore. Na druhej strane stoja jednoduché, no rýchle a praktické jednooknové dialógy bez širšieho vysvetlenia, čo návrhár realizuje. Títo sprievodcovia si vyžadujú lepšiu znalosť vzorov vo všeobecnosti ako i konkrétneho, práve aplikovaného, vzoru v porovnaní s viac dialógovými názornými sprievodcami.

Rozšírenie množiny návrhových vzorov

Je prirodzené, že množina návrhových vzorov sa rozširuje od oblasti k oblasti. Vznikajú nové vzory na všeobecnej úrovni ako i nové vzory v konkrétnych oblastiach (napr. objektovo – relačné mapovanie (Fowler, 2003)). Možnosť rozšírenia množiny návrhových vzorov podporovaných konkrétnym CASE nástrojom by mala byť skôr samozrejmosťou ako luxusom.

Na druhej strane v CASE nástroji musia byť vyriešené problémy spojené so zaznamenaním vzoru, parametrizáciou vzoru a uložením vzoru do interných štruktúr CASE nástrojov.

Podpora mikroarchitektúr

Mikroarchitektúry (Alur et al., 2001) sú definované v kontexte *enterprise* softvérových architektúr ako množiny spolupracujúcich vzorov pre riešenie rozsiahlejších problémov, pre realizáciu celých systémov alebo podsystémov.

Mikroarchitektúra je určená na riešenie hrubozrnných problémov, ktoré nemôžu byť vyriešené jediným vzorom. (Alur et al., 2001) Za príklad mikroarchitekúry môžeme považovať spoluprácu vzorov *Composite* a *Iterator* podľa obrázka 9-5, kedy *Iterator* zabezpečuje prechádzanie po štruktúrach uložených v inštancii triedy *Composite* vrapujúcej konkrétny kontajner.

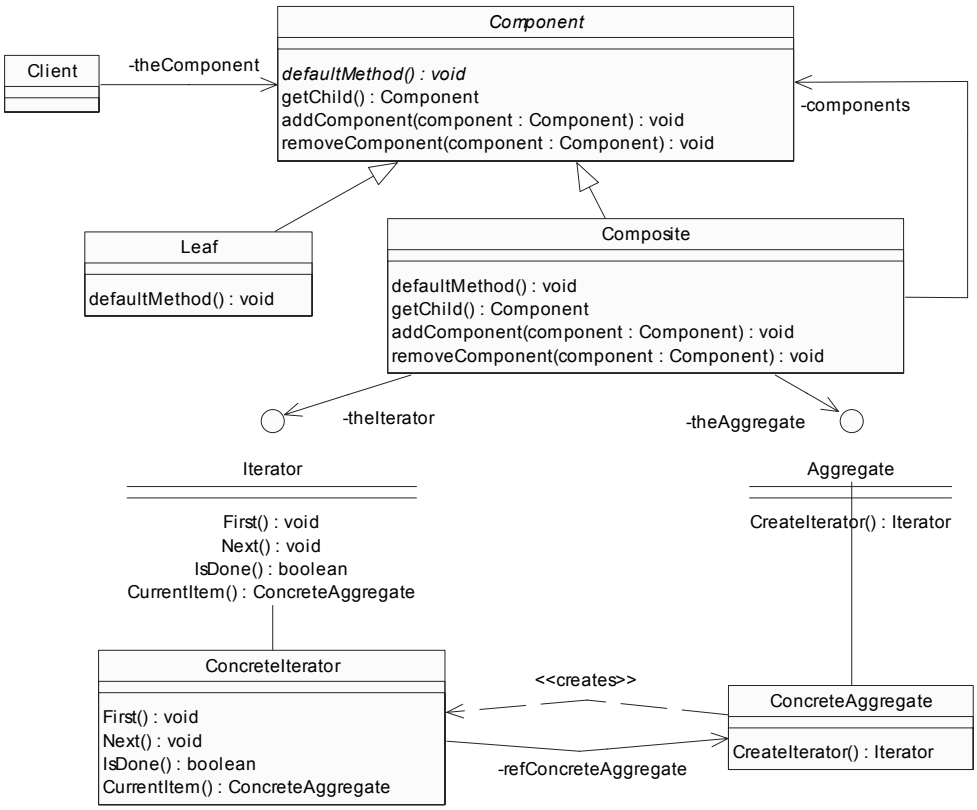
Vzor *Composite* rieši problém zovšeobecnenia zložení a jednotlivých prvkov a na sprístupňovanie jednotlivých prvkov vo vrapovanom kontajneri využíva možnosti vzoru *Iterator*. Návrhár používajúci vzor *Composite* navyše nemusí vedieť o používaní mikroarchitektúry so vzorom *Iterator*.

CASE nástroje by mali čo možno v najväčšej miere umožniť prácu s mikroarchitektúrami bez explicitnej znalosti návrhára.

Viditeľnosť vzoru v návrhu

Pri vytváraní rozsiahlych aplikácií je dôležité prehľadne udržiavať informácie o rozhodnutiach vo fáze návrhu. Jedným z rozhodnutí je určite aj rozhodnutie použiť ten ktorý návrhový vzor. Použitie návrhového vzoru v rozsiahlom návrhu by malo byť preto dostatočne viditeľné a zvýraznené najmä pre potreby neskoršej údržby.

CASE nástroje majú k dispozícii rôzne notácie (Dong, 2003, Jakubík, 2005a, Mak et al., 2004, Vlissides, 1998) a ďalšie prostriedky pre zaistenie tohto cieľa.



Obrázok 9-5. Mikroarchitektúra zložená zo vzorov Composite a Iterator.

9.3.2 Porovnanie vybraných CASE nástrojov podľa definovaných kritérií

Vzhľadom na podporu práce s navrhovanými vzormi sme porovnali 5 CASE nástrojov, pričom 2 z vybraných neobsahovali žiadnu podporu návrhových vzorov – išlo o Visual Studio 2005 verziu 8.0.50727.42 od Microsoftu a Enterprise Architect verziu 4.50.742 od Sparx Systems. Ďalšie z nástrojov Rational Rose verzia 2003.06.00.436.00, Rational XDE 1.5.0 oba nástroje od IBM a Together Designer od Borlandu podporovali viac alebo menej prácu s návrhovými vzormi. Tieto nástroje boli porovnané na základe definovaných kritérií.

Vývoj Rational Rose je v súčasnosti pozastavený a pokračuje v produktoch Rational XDE. Rational Rose podporuje prácu s 20 návrhovými vzormi podľa (Gamma et al., 1995), pričom využíva jednoduchých jednodialógových sprievodcov, generuje základné konštrukcie tried, metód a atribútov, pričom telá metód sú prázdne, resp. doplnené komentármi, podľa ktorých má vývojár postupovať. V prípade Rational Rose ide o základnú podporu návrhových vzorov.

Ďalej v tejto oblasti sa dostáva produkt Rational XDE, ktorý je pokračovateľom Rational Rose. Rozširuje podporu o pokročilých viacdialógových sprievodcov, ktorí vhodne informujú o obsadzovaní roly vo vzore, k dispozícii je dokumentácia vo forme helpu vzťahujúca sa k práve používanému vzoru, zdrojový kód je generovaný vo forme kostier tried, metód a atribútov. V niektorých prípadoch sa generujú i telá metód, ktoré

sú doménovo nezávislé. Rational XDE taktiež podporuje rozšírenie množiny používaných vzorov pomocou mechanizmu šablón.

Rovnakú úroveň podpory práce s návrhovými vzormi poskytuje i Together Designer od firmy Borland. Podobne ako Rational XDE disponuje pokročilými viacdialógovými sprievodcami pri vytváraní inštancií, generuje zdrojový kód s telami doménovo nezávislých metód. Navyše Together Designer má dobre prepracovanú možnosť rozšírenia množiny podporovaných vzorov, na základe modelov vzoru (napr. diagram tried) dokáže Together Designer získať potrebné informácie a pridať vzor do knižnice vzorov.

S vývojom odboru IT sa vyvíja aj podpora návrhových vzorov v CASE nástrojoch, v súčasnosti však žiaden z testovaných nástrojov nedisponuje prostriedkami pre znovupoužitie mikroarchitektúr (odhliadnuc od možnosti vytvorenia nového vzoru ako mikroarchitektúry) a možnosťou zviditeľniť návrhový vzor v komplexnom návrhu.

9.3.3 Sumarizácia porovnania CASE nástrojov a diskusia

Výsledky porovnania sú uvedené v tabuľke 9-3.CASE nástroje a ich podpora pre prácu s návrhovými vzormi predstavujú istú formu znovupoužitia návrhových vzorov.

Tabuľka 9-3. Sumarizácia porovnania CASE nástrojov.

	Rational Rose	Rational XDE	Borland Together
Podpora implementačných jazykov	C++, JAVA	C++, JAVA, .NET jazyky	JAVA, .NET jazyky
Podporované návrhové vzory	20 GoF vzorov	23 GoF vzorov	23 GoF vzorov
Generovanie zdrojového kódu	na základe požiadavky používateľa po vytvorení modelu	na základe požiadavky používateľa po vytvorení modelu	na pozadí vytvárania modelu
Vlastnosti generovaného zdrojového kódu	kostra projektu, tried, metód, atribútov	kostra projektu, tried, metód, atribútov spolu s časťami metód	kostra projektu, tried, metód, atribútov spolu s časťami metód
Vytváranie inštancie vzoru	jednoduchý sprievodca vo forme jedného dialógu	jedno/viacoknový sprievodca	multidialógový sprievodca s helpom vysvetľujúcim jednotlivé kroky
Rozšírenie množiny návrhových vzorov	nepodporuje	s použitím vzorových šablón, šablón zdrojového kódu	podpora pre vytvorenie vlastnej knižnice vzorov
Podpora mikroarchitektúr	nepodporuje	nepodporuje	nepodporuje
Viditeľnosť vzoru v návrhu	nepodporuje	nepodporuje	nepodporuje

Určite sa nedá povedať, že sme kritériami pokryli všetky smery podpory návrhových vzorov a taktiež ťažko hovoriť o reprezentatívnej vzorke porovnávaných nástrojov. Porovnaním práve vybraných nástrojov sme chceli poukázať na rozdelenie CASE nástrojov do troch skupín:

- nástroje bez podpory návrhových vzorov (napr. MS Visual Studio 2005),
- nástroje so základnou podporou návrhových vzorov (napr. Rational Rose),
- nástroje s rozšírenou podporou návrhových vzorov (napr. Borland Together).

Prvá skupina CASE nástrojov neponúkala žiadnu podporu v oblasti návrhových vzorov. Napriek tomu, že do tejto skupiny patria i pomerne používané nástroje, práve nepodporenie práce s návrhovými vzormi môže viesť k útlmu ich používania.

Do druhej skupiny CASE nástrojov patria buď staršie nástroje, ktorých nasledovníci sú súčasťou poslednej skupiny, alebo beta verzie nových nástrojov, ktoré majú ambície výraznejšie podporovať prácu s návrhovými vzormi či už v oblasti modelovania alebo generovania zdrojového kódu.

Do tretej skupiny patria nástroje spoločností, ktoré majú dlhodobé skúsenosti s modelovaním softvéru. Zdrojový kód po vygenerovaní obsahuje častokrát telá metód doplnené pomocnými komentármi pre ďalší vývoj. Do tejto skupiny tiež môžeme zaradiť nástroje zaoberajúce sa návrhovými vzormi ako takými, ktoré sú akýmisi mantinelmi určujúcimi ďalšie smerovanie v tejto oblasti. Nástroje v tejto skupine disponujú prepracovanou teóriou, ktorá určuje ďalší smer vývoja podpory práce s návrhovými vzormi.

Na druhej strane nemalým sklamaním je nepostačujúca podpora v oblasti viditeľnosti vzorov v rozšírenom návrhu, kedy už jednoduché označenie stereotypmi veľakrát nepostačuje, rovnako ako nepostačujúca podpora v oblasti práce s mikroarchitektúrami.

9.4 Využitie atypických jazykov pre dekompozíciu vzoru

Návrhové vzory sú súčasťou softvérového inžinierstva pomerne dlhú dobu, no podpora štandardne používanými objektovo-orientovanými jazykmi nie je príliš výrazná. Navyše samotné jazyky len v malej miere pomáhajú uľahčiť dekompozíciu vzoru. Štandardné objektovo-orientované jazyky v tomto ohľade dopĺňajú rôzne atypické a experimentálne jazyky, ktoré disponujú ďaleko širšou množinou prostriedkov použiteľných práve pre uľahčenie dekompozície vzoru. Medzi takéto jazyky môžeme zaradiť i jazyk Beta a s ním spojenú knižnicu DPL.

9.4.1 Špeciálne vlastnosti jazyka Beta

Jazyk Beta napriek svojmu veku disponuje v súčasnosti populárnymi prostriedkami ako jeden strom abstrakcie, virtuálne triedy alebo včlenené triedy. Dokonca v prípade včlenených tried (*nested classes*) sa stal vzorom pri vývoji jazyka JAVA.

V nasledujúcich častiach prejdeme niektoré význačné prvky ako i rozšírenia jazyka Beta použité pre vybudovanie a sprístupnenie knižnice vzorov aplikačnému vývojárovi.

Vzory

Jednou z charakteristických vlastností jazyka Beta je práve mechanizmus abstrakcie ako triedy, procedúry, metódy, ktoré sú unifikované v strome abstrakcie a sú odvodené od vzoru. Procedurálny vzor opisuje štruktúru aktivácie procedúry a môže byť použitý k parametrizácii tohto procesu, podobne vzor triedy opisuje štruktúru objektov a môže byť použitý na vytváranie objektov. V kontexte jazyka Beta teda môžeme hovoriť o vzoroch a subvzoroch. Pričom za subvzor považujeme vzor, ktorý dedí od vzoru umiestnenom vyššie v hierarchickom strome.

Tento mechanizmus jazyka Beta je podobný jednému stromu abstrakcie v súčasnosti používanom najmä v jazyku JAVA.

Virtuálne triedy

Virtuálne vzory v Bete môžu byť:

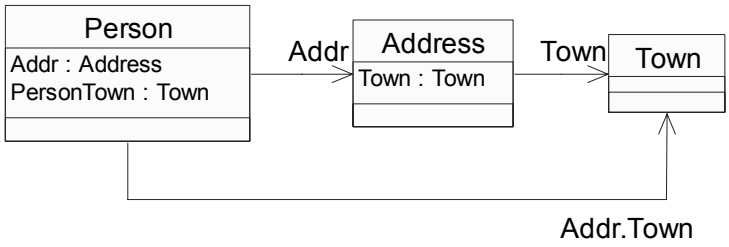
- vzor virtuálnej procedúry – korešponduje použitiu virtuálnej metódy v C++ alebo Smalltalk,
- vzor virtuálnej triedy – podobné parametrizovaným triedam ako generické triedy v Eiffel a šablóny v C++ alebo v JAVA.

Deklarácia atribútu *virtual* vo vzore znamená, že v čase deklarácie je známa len časť vlastností vzoru, plná špecifikácia je potom prenechaná na subvzor.

Premenovávanie

V článku (Madsen, 1992) je predstavený koncept premenovávania ako spôsob sprístupnenia vnorených atribútov ako by boli atribútmi pôvodnej triedy.

Na obrázku 9-6 trieda *Person* obsahuje atribút triedy *Address* a trieda *Address* obsahuje atribút triedy *Town*. Jednoduchou konštrukciou je potom možné priamo v triede *Person* sprístupniť atribút typu *Town*.



Obrázok 9-6. Použitie konceptu premenovávania v jazyku Beta.

Virtuálne neterminály

Beta poskytuje prostriedky ako napr. virtuálne neterminály pre rozšírenie spôsobu parametrizácie definovaných metód, resp. vzorov.

V príklade 9-2 je definovaný virtuálny neterminál pre príklad cyklu *while – loop*. Cyklus je potom parametrizovaný nasledovným zoznamom atribútov:

- podmienka prebiehania cyklu,
- spracovanie aktuálne prechádzaného elementu,

- podmienka pre opustenie cyklu,
- odkaz, kde má program pokračovať po skončení cyklu,
- spôsob výberu nasledujúceho prvku v cykle.

```
P: (#
     While:
     (# do
            Loop:
            (if <Cond: Evaluation>
            //True then
                   <PreImp: Imperative>;
                   (if <On: Evaluation>
                   //True then leave <Exit: label>
                   //False then
                          <PostImp: Imperative>;
                          restart Loop;
                   if)
            if)
     #)
#)
```

Príklad 9-2. Cyklus while-loop definovaný s použitím virtuálnych neterminálov.

V príklade 9-3 ďalej vidíme ako subvzor dodefinováva časti preddefinovaného cyklu. Pre dodefinovanie subvzor využíva použité virtuálne neterminály.

```
subP:P
     (#
            L:
            :
            .
            While‘: While
                   (#
                          Cond:: aList.NonEmpty;
                          PreImp:: aList.Current -> Key;
                          On:: Key = Subject
                          Exit:: L
                          PostImp:: aList.Next
                   #)
     #)
```

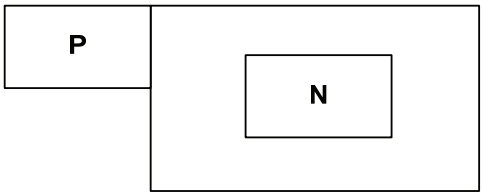
Príklad 9-3. Naviazanie virtuálnych neterminálov na časti zdrojového kódu.

Autori sa v (Agerbo & Cornils, 1997) zaväzujú používať notáciu OMT (*Object Modeling Technique*) (predchodcu súčasne používaného UML), no vzhľadom na možnosti jazyka Beta im táto notácia nepostačuje, a tak navrhli rozšírenia OMT pre potreby zápisu modelov implementovaných v jazyku Beta pomocou špeciálnych konštrukcií jazyka Beta. Ďalej si uvedieme len niektoré základné rozšírenia.

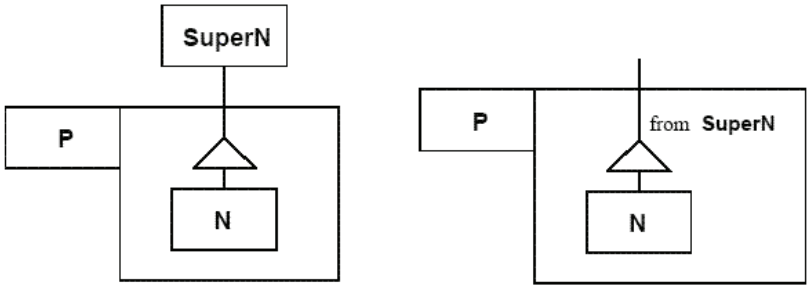
Včlenené triedy

Podobne ako v jazyku JAVA existujú vnorené triedy, v jazyku Beta sú k dispozícii včlenené triedy, ktoré nie sú viditeľné mimo nadradenej triedy, no môžu využívať akékoľvek dedenia od iných tried.

Na obrázku 9-7 je uvedený príklad včlenenej triedy N do triedy P. Na obrázku 9-8 sú uvedené alternatívy zápisu dedenia včlenenej triedy N od triedy SuperN.



Obrázok 9-7. Príklad včlenenej triedy N do triedy P.

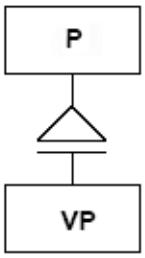


Obrázok 9-8. Dedenie v prípade včlenenej triedy.

Virtuálne triedy

Jazyk Beta disponuje tzv. virtuálnymi triedami, ktoré sa podobajú generickým triedam z jazyka Eifel alebo šablónam v jazyku C++, JAVA. Vzhľadom na zápis v OMT bolo potrebné navrhnúť rozšírenie i pre zápis virtuálnych tried.

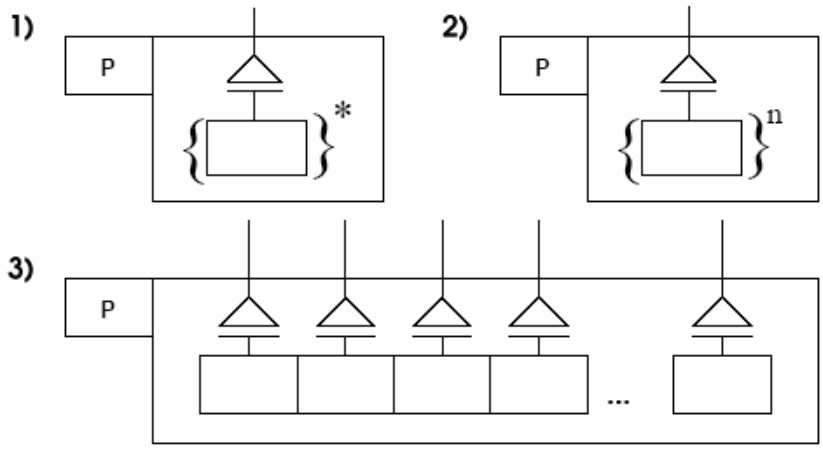
Na obrázku 9-9 uvádzame zápis virtuálnej triedy VP, ktorá je parametrizovaná triedou P.



Obrázok 9-9. Zápis virtuálnej triedy VP parametrizovanej triedou P.

Zoznamy tried

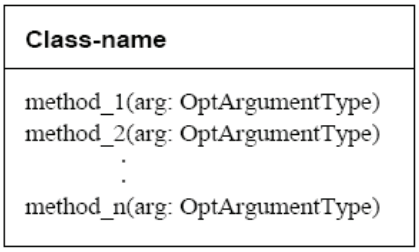
V jazyku Beta je povolené v rámci jedného vzoru udržiavať zoznam včlenených tried. Táto vlastnosť je simulovaná zoznamom včlenených virtuálnych tried. Pre zápis tohto zoznamu tried autori použili rozšírenie OMT uvedené na obrázku 9-10, pričom hviezdička, resp. n znamená obmedzenie na počet vnorených virtuálnych tried.



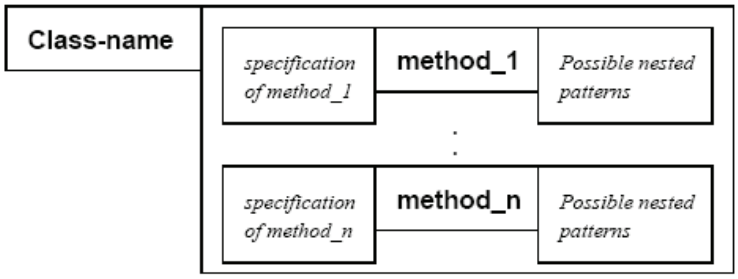
Obrázok 9-10. Rôzne možnosti zápisu zoznamov tried v jazyku Beta.

Vzor metódy

V jazyku Beta je metóda definovaná ako včlenený vzor do iného vzoru. Konkrétne Vzor metódy je včlenený do Vzoru triedy. Na rozdiel od štandardného zobrazenia metód v OMT (obrázok 9-11) autori rozšírili notáciu o zápis vzoru metódy včleneného do vzoru triedy (obrázok 9-12).



Obrázok 9-11. Štandardný zápis metód v rámci notácie OMT.



Obrázok 9-12. Zápis vzoru metódy včleneného do vzoru triedy.

Autori rozšírení sa však obmedzili na zápis metód pomocou vzorov iba v nevyhnutných prípadoch, kedy bude vzťah metódy k vzoru metódy nevyhnutný k pochopeniu modelu.

9.4.2 Analýza vzorov v (Agerbo & Cornils, 1997) a nadväznosť na jazyk JAVA

Autori v (Agerbo & Cornils, 1997) analyzujú všetkých 23 vzorov podľa (Gamma et al., 1995) pričom návrhové vzory porovnávajú ku konštrukciám jazyka. Vychádzajú z tvrdenia uverejneného v (Gamma et al., 1995), kde autori tvrdia, že návrhový vzor jedného vývojára môže byť iba primitívnym stavebným blokom iného vývojára. Analýzy vyúsťujú do výsledkov zhrnutých v tabuľke 9-4.

Tabuľka 9-4. Klasifikácia vzorov na idey a idiómy.

Návrhový vzor	*Klasifikácia*	*Podpora jazyka*
Abstract Factory	idea	
Builder	idea	
Factory Method	idióm	Virtuálne triedy
Prototype	idióm	Premenné vzoru
Singleton	idióm	Singulárne objekty
Adapter	idióm	Explicitná delegácia
Bridge (v čase kompilácie)	idióm	Fragment jazyka
Bridge (počas vykonávania)	idea	
Composite	idea	
Decorator	idea	
Facade	idióm	Včlenené triedy
Flyweight	idea	
Proxy	idióm	Explicitná delegácia
Chain of responsibility	idióm	Explicitná delegácia
Command (enkapsulácia metódy)	idióm	Vzor procedúry
Command (odpojenie iniciátora od interpreta)	idea	
Interpreter	idea	
Iterator	idea	
Mediator	idea	
Memento	idea	
Observer	idea	
State	idea	
Strategy	idea	
Visitor	idióm	virtuálne neterminály
Template method	idióm	viacnásobná správa

V tabuľke 9-4 sú vzory klasifikované na tzv. idiómy a idey. Pričom pod pojmom idióm rozumieme návrhový vzor, ku ktorému existuje v jazyku Beta priama podpora.

K jednotlivým idiómom sú uvedené aj konštrukcie jazyka použité pre ich pokrytie. Ostatné návrhové vzory sa nedajú jednoducho podporiť priamo prostriedkami jazyka Beta a teda je potrebné použiť viacero konštrukcií, aby sme vzor pokryli pomocou prostriedkov jazyka. Takéto návrhové vzory sú označené ako idey.

Autori identifikovali a vybrali konkrétne prostriedky jazyka Beta ku každému z idiómov. Jazyk Beta slúžil ako inšpirácia pri definovaní súčasne používaných objektovo-orientovaných jazykov ako C++ a JAVA. Dokonca v prípade jazyka Java autori priamo prebrali niektoré z myšlienok jazyka Beta. V tabuľke 9-5 uvádzame mapovanie idiómov na prostriedky jazyka JAVA spolu s podporou v jazyku Beta. Ide o transformáciu názvoslovia z jedného jazyka do druhého a opačne. V niektorých prípadoch je doplnená konkretizácia rozdielov jednotlivých prostriedkov.

Tabuľka 9-5. Mapovanie idiómov na prostriedky jazyka JAVA.

Návrhový vzor	*Podpora v jazyku JAVA*	*Ekvivalent v jazyku Beta*
Factory Method	šablóny, generické triedy	Virtuálne triedy
Prototype		Premenné vzoru
Singleton		Singulárne objekty
Adapter		Explicitná delegácia
Bridge (v čase kompilácie)		Fragment jazyka
Facade	Vnorené triedy ale i verejné rozhrania a triedy, ktoré sprístupňujú funkcionalitu celého balíka tried.	Včlenené triedy
Proxy		Explicitná delegácia
Chain of responsibility		Explicitná delegácia
Command (enkapsulácia metódy)		Vzor procedúry
Visitor		virtuálne neterminály
Template method		viacnásobná správa

Napríklad včlenené triedy pri vzore *Facade* sú v jazyku Java nazývané ako vnorené triedy. Vnorené triedy ako také nie sú prístupné pre vonkajšie triedy. Sú prístupné v hlavnej triede, podľa ktorej je nazvaný súbor a sú prístupné pre všetky ostatné triedy definované v súbore hlavnej triedy. Pri týchto obmedzeniach sa dá povedať, že hlavná trieda predstavuje fasádu pre systém vnorených tried, resp. systém včlenených tried. Tento príklad sa dá namapovať i na skupinu tried zaobalených v balíku, pričom existujú verejné triedy alebo rozhrania, pomocou ktorých je funkcionalita balíka sprístupnená vonkajším balíkom. Pri týchto obmedzeniach opäť môžeme hovoriť

o fasáde, ktorú predstavujú verejné triedy a verejné rozhrania, pričom subsystém tried zaobalený v balíku je pomocou fasád oddelený od ostatných subsystémov.

9.4.3 Knižnica návrhových vzorov LDP

Autori sa vzhľadom na prostriedky ponúkané jazykom Beta rozhodli pre zabezpečenie znovupoužitia častí návrhových vzorov formou knižnice vzorov LDP. Knižnica LDP sústreďuje znovupoužiteľné časti návrhových vzorov, ktoré sú v aplikácii ohýbané rôznymi prostriedkami. Parametrizácia vzorov je z veľkej miery zabezpečená virtuálnymi neterminálmi, virtuálnymi triedami a ďalšími možnosťami jazyka Beta. Autori sa nevyhli použitiu viacnásobného dedenia, ku ktorému ale uviedli alternatívu vo forme použitia kompozície.

Vhodnosť výberu správneho jazyka sa prejaví najmä pri znovupoužívaní častí návrhových vzorov. Knižnica LDP obsahuje niekoľko návrhových vzorov podľa (Gamma et al., 1995) a poukazuje na možnosť vytvorenia takejto knižnice. Cieľ v podobe vytvorenia prostriedku pre znovupoužitie návrhových vzorov bol naplnený, no nájsť riešenie využívajúce túto knižnicu je ťažké, ak vôbec nejaké existuje.

Použitie takejto knižnice vzorov so sebou prináša výhody i nevýhody. Medzi výhody určite patrí možnosť znovupoužia častí vzorov, pričom nie je potrebné zaoberať sa implementáciou opakujúcich sa častí. Ďalšou z výhod je možnosť výraznejšej dokumentácie a priradeniu tried k vzorom. Jednoduchým spôsobom môžeme identifikovať participantov v inštancii konkrétneho vzoru. Dokumentácia môže byť generovaná poloautomatickým spôsobom.

K nevýhodám patrí nemožnosť modifikovať, resp. ohýbať vzor iným, ako v knižnici definovaným (naznačeným a povoleným) spôsobom. Vzor je uložený v knižnici a sprístupnené sú len jeho časti, pričom aj tie je možné modifikovať iba spôsobmi, ktoré poskytuje konkrétny implementačný jazyk. Gamma vo svojej monografii (Gamma et al., 1995) v prípade viacerých vzorov opísal rôzne spôsoby ohýbania vzoru. Istá možnosť ako sprístupniť viacero smerov v ohýbaní vzoru je v umiestení do knižnice viacero kópií vzoru, pričom v každej kópii vzoru bude povolený iný smer ohýbania. Ale opäť sa tým znižuje možnosť a voľnosť vlastnej modifikácie vzoru podľa potreby konkrétnej aplikácie.

Na druhej strane, ak identifikujeme základné spôsoby ohýbania vzoru, môžeme vylúčiť chyby, ktoré môžu nastať nepovolenou a nesprávnou modifikáciou vzoru. Ďalšou z nevýhod môže byť prípadná nutnosť použitia všeobecného názvoslovia, ktoré síce vo viacerých spôsoboch vyjadruje jasnú príslušnosť k tomu ktorému vzoru, no na druhej strane v prípade konkrétnej domény môže byť príliš všeobecné a niekedy až mätúce.

Autorom sa podarilo preskúmať závislosti u všetkých 17 vzorov, ktoré neoznačili za idiómy s výnimkou *Adapter*, *Proxy* a *Chain of responsibility*. Výsledky vyhodnotenia možností uloženia vzoru do knižnice sú zobrazené v tabuľke 9-6.

Hlavným dôvodom neuloženia niektorých vzorov do knižnice LDP je veľkosť všeobecnej časti konkrétneho vzoru. Táto veľkosť bola príliš malá a znovupoužitie by bolo zanedbateľne malé. Aj napriek výrazne malému znovupoužitiu, koncentrovaním niektorých vzorov, resp. ich častí do knižnice by sa podporila viditeľnosť v celkovom návrhu.

Vzhľadom na mechanizmy abstrakcie poskytnuté jazykom Beta, najviac boli využívané virtuálne zoznamy s dynamickými listami v prípade *Builder*, *Bridge*, *Composite*, *Chain of responsibility*, *Flyweight*, *Iterator* ale tiež virtuálne triedy v prípade *Bridge*, *Flyweight*, *Command* a *Observer*. Ako príklad na obrázku 9-13 uvádzame rozdelenie vzoru *Composite* na časť umiestnenú v aplikácii a všeobecnú časť umiestnenú v knižnici. Diagram je znázornený v notácii OMT.

Kde 1 predstavuje operáciu s cyklom, ktorá pre každý element uložený v kontajneri zavolá operáciu s názvom *Operation*, podľa príkladu 9-4.

```
for all c in Children
do c.Operation(op)
```

Príklad 9-4. Cyklické volanie Operation pri Composite.Operation.

Tabuľka 9-6. Možnosti znovupoužitia vzoru formou knižnice.

Návrhový vzor	*Možnosť využitia v kontexte knižnice*
Abstract Factory	
Builder	Áno
Adapter	
Bridge (runtime)	Áno
Composite	Áno
Decorator	Áno
Flyweight	Áno
Proxy	
Chain of responsibility	Áno
Command (odpojenie iniciátora od interpreta)	Áno
Interpreter	Áno
Iterator	Áno
Mediator	
Memento	
Observer	Áno
State	
Strategy	

Body 2, 3 a 4 na obrázku 9-13 predstavujú volanie triedy *Composite,* resp. jej metódy *Operation* vrapujúcej konkrétny kontajner s názvom doménovej operácie podľa príkladu 9-5, ktorá sa má vykonať nad jednotlivými prvkami kontajneru ako parameter operácie *Operation*.

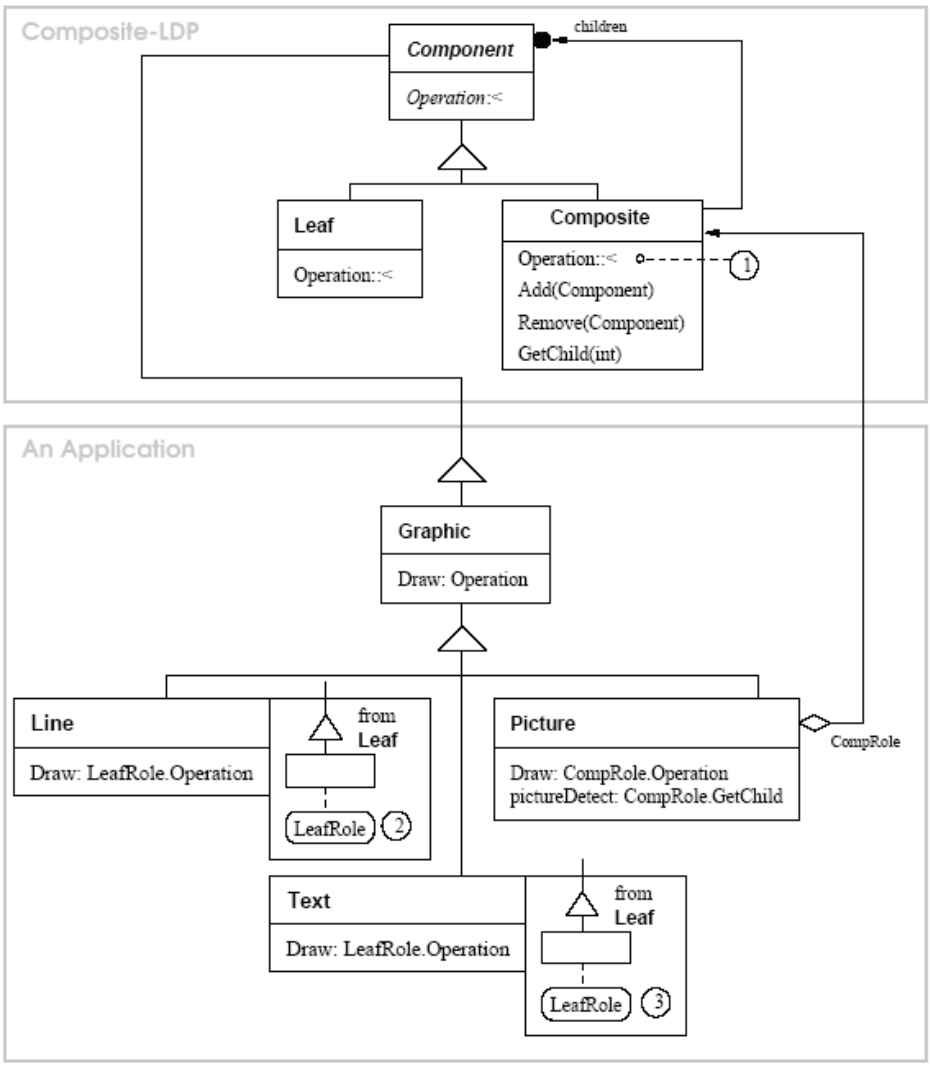
```
CompRole.Operation(draw)
CompRole.Operation(delete)
```

Príklad 9-5. Volanie všeobecnej Operation s názvom doménovo závislej funkcie.

Metóda *Operation* označená ako 7 v triedach *Line*, *Text* a *Picture* rozhoduje o vykonaní konkrétnych doménovo závislých metód na základe parametra podľa príkladu 9-6.

```
if op = draw then Draw()
else if op = delete then Delete()
```

Príklad 9-6. Vykonanie doménovo závislých operácií podľa vstupného parametra.



Obrázok 9-13. Návrhový vzor Composite rozdelený na všeobecnú a aplikačnú časť.

9.4.4 Diskusia

Napriek tomu, že jazyk Beta a tiež knižnica DPL nepatria k novinkám v objektovo-orientovanom programovaní, ide v prípade DPL o jedinečné riešenie v podpore znovupoužitia návrhových vzorov na úrovni zdrojového kódu. Vzhľadom na pomerne slušné zdokumentovanie rozhodnutí pri dekompozícii jednotlivých návrhových vzorov, môže táto práca slúžiť ako inšpirácia pri migrácii knižnice DPL na inú platformu, iný programovací jazyk.

Jazyk Beta poskytol svojimi prostriedkami a možnosťami výbornú základňu pre vývoj knižnice DPL. Škoda len, že viaceré prostriedky neboli dostatočne inšpiratívne ako napríklad včlenené triedy a nestali sa súčasťou aktuálne používaných jazykov.

V mnohých prípadoch zrejme ide o príliš konkrétne prostriedky a nástroje, ktoré by vniesli vyššiu zložitosť do jazyka a výsledný efekt by ani zďaleka nezodpovedal vynaloženému úsiliu. Na druhej strane v prípade jazyka Beta ide skôr o experimentálny ako komerčne vyvíjaný a podporovaný jazyk.

Súčasný vývoj zameraný alebo ovplyvnený jazykom JAVA sa zaoberá inými možnosťami rozšírenia samotného jazyka. Tieto rozšírenia sú realizované rôznymi spôsobmi s využitím všetkého čo JAVA ponúka a častokrát v mnohom pripomínajú niektoré z prostriedkov jazyka Beta.

9.5 Použitie štandardných jazykov pre dekompozíciu vzoru

Jednou z ďalších možností ako zabezpečiť rozdelenie návrhového vzoru na všeobecnú a doménovo závislú časť je využitie prostriedkov objektovo-orientovaných jazykov a izolácia všeobecnej časti vzoru do knižnice. V článku (Jakubík, 2005b) sme sa zaoberali rozdelením návrhového vzoru *Composite* s použitím prostriedkov jazyka C++. Tento článok bol zároveň hlavným zdrojom pre nasledujúce časti.

9.5.1 Dekompozícia vzoru Composite s využitím C++

V nasledujúcich častiach sa pokúsime rozdeliť obe implementácie vzoru *Composite* (bezpečnú aj transparentnú) na časť všeobecnú a časť doménovo závislú s použitím prostriedkov jazyka C++.

Ďalej budeme pracovať s triedami návrhového vzoru *Composite* pričom rozdelíme triedy *Component* a *Composite* na *CommonComponent*, *ConcreteComponent* a *CommonComposite*, *ConcreteComposite*, kde triedy s prefixom *Common* predstavujú všeobecné časti vzoru a triedy s prefixom *Concrete* doménovo závislé časti vzoru. V oboch modeloch implementácie vzoru *Composite* nenájdete všeobecnú variantu triedy *Leaf*, pretože trieda *Leaf* je doménovo závislá vo všetkých smeroch a v modeloch sa vyskytuje pod názvom *ConcreteLeaf*.

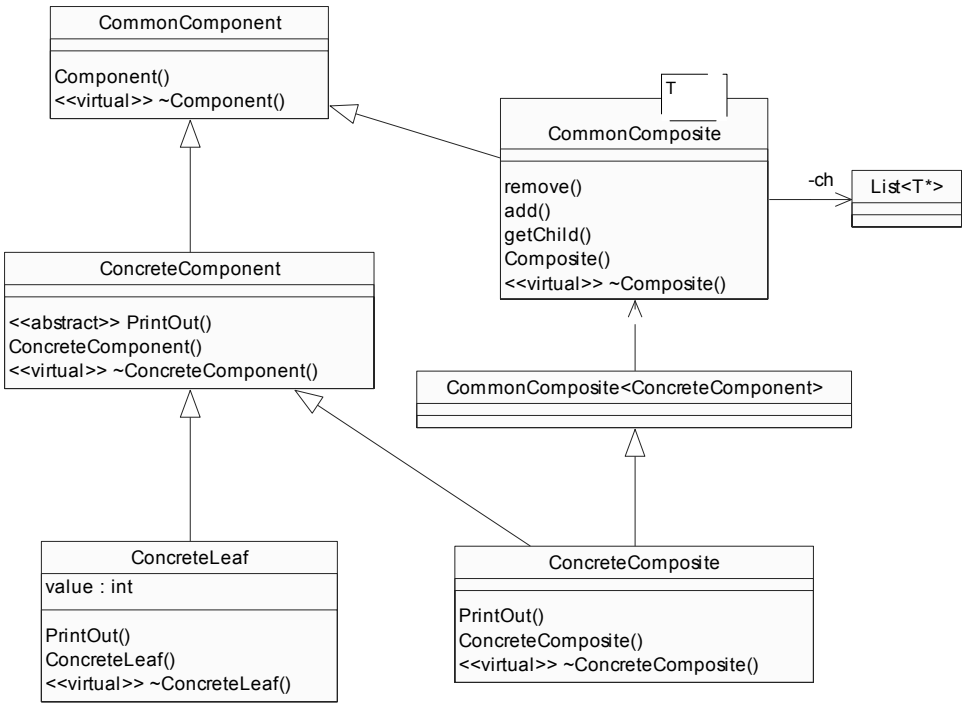
CommonComposite obsahuje podporu pre kontajner manažment ako je výber, odstraňovanie a pristupovanie k prvkom kontajnera. *ConcreteComposite* zabezpečuje podporu doménovo špecifických operácií. Implementácia triedy *CommonComponent* je závislá od konkrétneho typu vzoru *Composite*, môže byť prázdna pri bezpečnej skladbe alebo môže definovať operácie spojené s kontajner manažmentom pri transparentnej skladbe.

Doteraz sme iba málo hovorili o konkrétnom jazyku implementácie. Je ale dôležité vedieť, aké prostriedky môžeme v nasledujúcich experimentoch použiť. Použijeme C++ s možnosťami viacnásobného dedenia a mechanizmom šablón pre ilustráciu komplexnosti a netriviálnosti problému.

9.5.2 Dekompozícia vzoru pri bezpečnej skladbe s využitím C++

Pri bezpečnej skladbe nie je nutné definovať žiadnu operáciu v triede *CommonComponent*. V procese návrhu knižnice nepoznáme doménovo závislé operácie. Tieto operácie sme oddelili do triedy *ConcreteComponent*, ktorá dedí od triedy *CommonComponent*. *ConcreteComponent* potom predstavuje rozhranie, ktoré musí byť implementované všetkými listami i triedou *ConcreteComposite*. Navyše trieda *ConcreteComposite* musí dediť operácie spojené s kontajner manažmentom od *CommonComposite*.

CommonComposite sme parametrizovali *ConcreteComponent*-om pre došpecifikovanie operácií spojených s kontajner manažmentom. Pomocou tejto parametrizácie sme vyriešili aj chýbajúcu asociáciu medzi triedou *Composite* a *Component* definovanú vzorom *Composite*. Pre parametrizáciu *CommonComposite* sme využili mechanizmus šablón. Schéma riešenia je zobrazená formou diagramu tried na obrázku 9-14, na obrázku je zvýraznená znovupoužiteľná časť vzoru, ktorá je umiestnená v knižnici.



Obrázok 9-14. Diagram tried predstaveného riešenia pre bezpečnú skladbu.

Vzor sme rozdelili na všeobecnú a doménovo závislú časť. Všeobecnú časť predstavujú triedy *CommonComponent*, *CommonComposite* a *List*. Trieda *CommonComposite* zaobaľuje operácie nad *List*-om. Triedy *CommonComposite* a *List* sú implementované pomocou šablón, proces vytvárania inštancií je parametrizovaný konkrétnym komponentom v našom prípade *ConcreteComponent.*

Konkrétnu časť vzoru predstavujú triedy *ConcreteComponent*, *ConcreteLeaf* a *ConcreteComposite*. Trieda *ConcreteComposite* je vytváraná viacnásobným dedením od triedy *ConcreteComponent*, od ktorej dedí doménovo závislé operácie a od triedy *CommonComposite*, od ktorej dedí operácie spojené so zaobaľovaním triedy List.

9.5.3 Dekompozícia vzoru pri transparentnej skladbe s využitím C++

Rozdiel medzi bezpečnou a transparentnou skladbou je v umiestnení definícií metód spojených s kontajner manažmentom. V prípade transparentnej skladby sú metódy definované už v triede *Component*. Metódy súvisiace s kontajner manažmentom musia byť implementované v každom liste a tiež v triede *Composite*.

Vo všeobecnej časti vzoru musíme modifikovať triedu *CommonComponent*, ktorá definuje rozhranie (spolu s preddefinovanou implementáciou) metód zabezpečujúcich kontajner manažment. Metódy definované v tejto triede musia byť prekryté v *CommonComposite* a môžu byť prekryté (podľa typu implementácie v *Common-Component*) v každom liste. Štruktúra riešenia je znázornená na obrázku 9-15, na obrázku je zvýraznená znovupoužiteľná časť vzoru, ktorá je umiestnená v knižnici.

Triedu *CommonComponent* sme vytvorili ako šablónu, ktorá je parametrizovaná typom *ConcreteComponent* pre dodefinovanie metód spojených s kontajner manažmentom. Tieto metódy môžu obsahovať default implementáciu alebo môžu byť čisté virtuálne metódy bez akejkoľvek implementácie. V našom experimente sme použili čisté virtuálne metódy a definovali sme konkrétne správanie v *CommonComposite* (štadardne pre pridávanie, odoberanie a sprístupňovanie prvkov v použitom kontajneri) a v *ConcreteLeaf* (*getChild* – s návratovou hodnotou null, a ostatné metódy s prázdnou implementáciou).

Trieda *ConcreteLeaf* dedí od triedy *ConcreteComponent*, ktorá dedí od *Common-Component* s definíciou metód pre kontajner manažment. Tento model implementácie umožňuje implementovať metódy spojené s kontajner manažmentom už v triede *ConcreteComponent* alebo v triedach *ConcreteLeaf* podľa potreby.

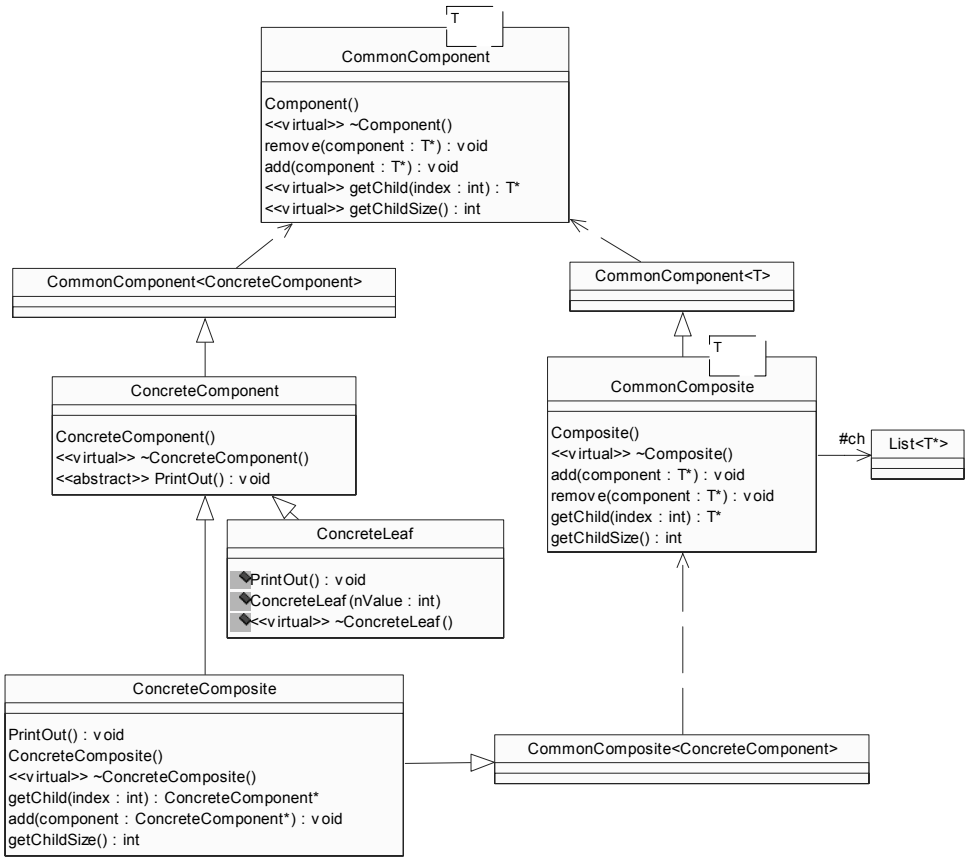
Trieda *ConcreteComposite* využíva viacnásobné dedenie pre zloženie správania súvisiaceho s kontajner manažmentom z *CommonComposite* a doménovo závislého správania z *ConcreteComponent*.

Vzťah medzi triedami *Component* a *Composite* zo vzoru *Composite* je v tomto prípade implementovaný cez šablónu *CommonComposite*, ktorá je parametrizovaná doménovo závislým typom *ConcreteComponent*.

9.5.4 Diskusia

Predstavené riešenie dekompozície vzoru *Composite* v jazyku C++ nie je jediné možné, no už pri prvom pohľade je vidieť čiastočnú komplexnosť a náročnosť riešenia. Použitie šablón a viacnásobného dedenia dáva jasné obmedzenia na implementačný jazyk.

Aj keď použitie všeobecných častí v aplikácii by jasne určilo príslušnosť tried ku vzoru, do aplikácie by sa vniesla zbytočná komplexnosť. Navyše rozdelenie tried na všeobecné a doménovo závislé narúša ku vzoru *Composite* zodpovedajúci model rolí, čo súvisí s komplikovaním riešenia a zvyšovaním neprehľadnosti.



Obrázok 9-15. Diagram tried predstaveného riešenia pre transparentnú skladbu.

Otázka na druhú stranu znie, či je vôbec možné znovupoužiť časti vzoru bez fyzického rozdelenia tried spájajúcich všeobecnú a doménovo závislú časť vzoru?

Ako uvidíme v časti 9.6, je to možné s pomocou špeciálnych rozšírení. Táto časť mala za cieľ poukázať na netriviálnosť realizácie dekompozície vzoru pomocou štandardných prostriedkov jazyka C++.

Navyše sa podarilo poukázať na nutnosť prijatia porušení v obmedzeniach modelu rolí v kontexte návrhového vzoru *Composite* s cieľom oddeliť všeobecnú časť od doménovo závislej časti vzoru (napr. v podobe obsadenia role *Component* triedami *ConcreteComponent* a *CommonComponent*, čím sa porušili obmedzenia na počet tried obsadených v danej roli).

9.6 Rozšírenie množiny prostriedkov jazyka JAVA

Jedným v súčasnosti z najčastejšie používaných objektovo – orientovaných jazykov je jazyk JAVA. Podpora návrhových vzorov v základnej verzii jazyka ani zďaleka nepokrýva potreby pre znovupoužitie návrhových vzorov, aj keď sa dá povedať, že niektoré z návrhových vzorov sú súčasťou i základných knižníc jazyka, a teda sú používané bez explicitných znalostí o ich štruktúre. Ide však len o jednoduché vzory, ktoré sú už takmer chronicky známe napr. Iterator alebo Observer.

Jazyk JAVA prebral isté vlastnosti z jazyka Beta, detailnejšie sú tieto vlastnosti opísané v časti 9.4 v analýze prostriedkov jazyka Beta. Jazyk JAVA umožňuje rôzne spôsoby jeho rozšírenia napr. s využitím reflexie. Autori článku (Lahire & Quintian, 2003) identifikujú isté možnosti rozšírenia jazyka JAVA, pričom tieto rozšírenia nie sú priamo určené pre podporu návrhových vzorov, ale pre podporu znovupoužitia v objektovo – orientovaných jazykoch všeobecne. Jednotlivé rozšírenia jazyka pre podporu znovupoužitia sú navrhnuté tak, aby kládli čo najmenšie nároky na implementačný jazyk. Testovací prototyp je vytvorený pre jazyk JAVA a v kontexte tohto jazyka boli vyberané i identifikátory novodefinovaných operácií a atribútov. Názvoslovie je príbuzné s jazykom JAVA, pričom priamo jazyk JAVA je použitý iba v častiach opisujúcich konkrétnu funkcionalitu daného elementu.

V nasledujúcich častiach prejdeme niektoré z rozšírení a uvedieme ich v kontexte znovupoužitia návrhových vzorov. Cieľom opäť bude oddeliť všeobecné časti vzoru do knižnice a pomocou novonavrhnutých prostriedkov ich sprístupniť v konkrétnej aplikácii. Autori v článku uvádzajú príklad ohýbania vzoru *Observer* do aplikácie s jednoduchým grafickým rozhraním.

9.6.1 Koncerny

Vo všeobecnosti môžeme za koncern považovať jeden fazet (rez) problematiky, ktorú rieši aplikácia. Koncern je teda zložený z entít (triedy, rozhrania a pod.), resp. z hierarchií entít zaobalených v jednom kontajneri nazývanom balík, ktorý prípadne môže obsahovať ďalšie sub – kontajnery. Koncern pripomína štruktúru balíkov v JAVA, ale nemusí s nimi implicitne súvisieť.

Koncerny, v závislosti od ich charakteristiky, môžu byť:

- funkcionálne,
- nefunkcionálne,
- hybridné.

Aplikácia je vo väčšine prípadov zložená z viacerých fazetov. Každý z fazetov korešponduje s množinou funkcionalít. Časť softvéru implementujúca konkrétnu funkcionalitu je označovaná ako funkcionálny koncern. Napríklad knižnice tried poskytované objektovo-orientovanými jazykmi a určené k jedinému cieľu môžu byť označené ako koncern.

Ak aplikácia potrebuje rozšíriť existujúcu funkcionalitu s ďalším spracovaním ako je napr. udržiavanie perzistencie dát, distribúcia dát, podpora rôznych druhov bezpečnosti ide o nefunkcionálny koncern.

Hybridný koncern nielenže rozširuje už existujúcu funkcionalitu (čím ho môžeme považovať za nefunkcionálny koncern), ale navyše pridáva i novú funkcionalitu (a mohli by sme ho považovať za funkcionálny koncern).

V nezávislosti, či ide o funkcionálny alebo nefunkcionálny koncern, môže byť obsah koncernu v aplikácii umiestnený typicky dvomi spôsobmi:

- na jedinom mieste (trieda, hierarchia tried),
- rozšírený po celom systéme tried.

Súčasné znovupoužitie v objektovo-orientovaných jazykoch je založené na použití dedenia a asociácií, čo však vedie k drahej modifikácii výsledných vlastností záverečného produktu. Oddelenie častí produktu na základe konkrétneho konceptu je náročné a pomocou prostriedkov, ktoré poskytujú súčasné objektovo-orientované jazyky neprehľadné a príliš komplexné.

Autori sa rozhodli vyriešiť rozptýlenie koncernu v rôznych triedach hierarchie s využitím implementačne nezávislého prostriedku tzv. adaptácií.

9.6.2 Typy adaptácie ako rozšírenia jazyka

Ako sme spomenuli skôr, koncern je zaobalený v hierarchii kontajnerov, ktoré obsahujú klasifikátory. Klasifikátor prislúcha ku kontajneru a má modifikátory (*abstract*, *final*, *public*). Obsahuje atribúty (konštanty, inštancie, premenné triedy) a metódy (procedúry, funkcie, konštruktory). Metódy majú signatúru (návratový typ, parametre, názov metódy), modifikátory a samotné telo s funkcionalitou.

Adaptácia ako taká je zahrnutá v konkrétnom adaptéri, resp. v skupine adaptérov a je teda fyzicky oddelená od triedy, na ktorú sa vzťahuje. Každý adaptér má svoje jedinečné meno, podľa ktorého môže byť identifikovaný. Môže byť abstraktný alebo konkrétny a môže dediť správanie od ďalších adaptérov.

Adaptér obsahuje ciele adaptácie, ktoré môžu byť abstraktné alebo konkrétne v závislosti od entít, na ktoré sa vzťahujú. Cieľ adaptácie je naviazaný na klasifikátor, metódu alebo atribút. V prípade abstraktného cieľa adaptácie je možné určiť isté obmedzenia na cieľ adaptácie. V prípade, že ide o konkrétny cieľ adaptácie, cieľ obsahuje konkrétny zoznam identifikátorov (v závislosti na type, názvu klasifikátora, metódy alebo atribútu). Adaptér tiež obsahuje operátory adaptácie. Poznáme dva typy adaptérov:

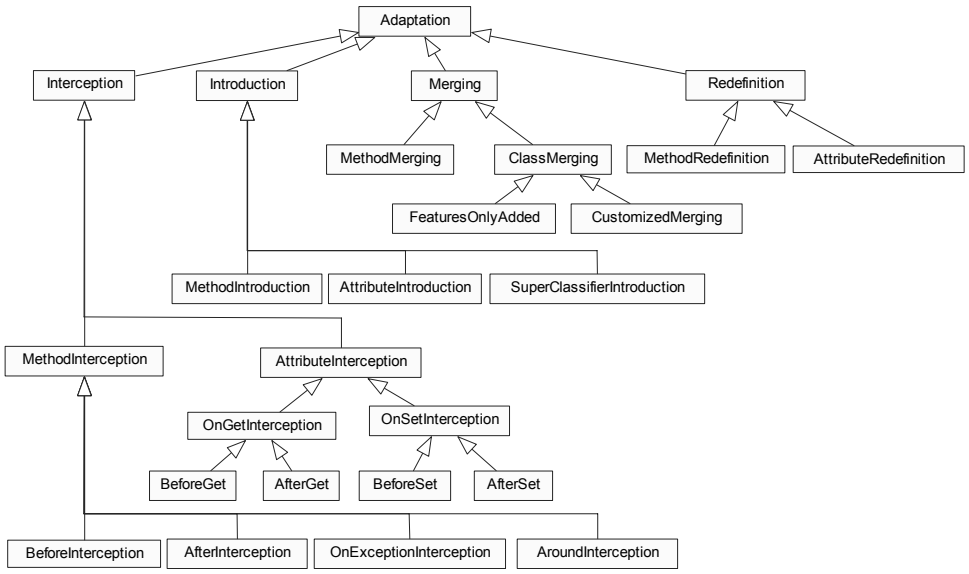
- ex-situ,
- in-situ.

Ak je adaptér definovaný ako ex-situ, potom je vytvorený nový kontajner a koncern spojený s adaptáciou je vložený do neho. Ak ide o in-situ adaptér, potom sa využíva už existujúci kontajner a adaptácia prebieha priamo nad ním.

V článku (Lahire & Quintian, 2006) navrhnutý model podporuje 6 nižšie uvedených kategórií adaptácií, pričom každá z nich obsahuje viacero typov konkrétnych adaptácií. Adaptácie v kontexte modelu sú uvedené na obrázku 9-16.

Jednotlivé adaptácie patria vždy do jednej z kategórií adaptácií:

- implementácia nových rozhraní, vloženie novej super triedy (*SuperClasifierIntroduction*),
- fúzia metód (*MethodMerging*) s viacerými variantmi napr. *MergingBefore*, *MergingAfter,*
- vloženie (*MethodIntroduction*) alebo redefinícia metód (*MethodRedefinition*),
- vloženie novej premennej do triedy (*AttributeIntroduction*),
- zachytenie pred (*BeforeInterception*), po (*AfterInterception*), okolo (*AroundInterception*) konkrétnej udalosti, zachytenie výskytu výnimky (*OnExceptionInterception*),



Obrázok 9-16. Podporované typy adaptácie v kontexte modelu rozšírení.

- zachytenie prístupu k inštancii alebo k premennej triedy (*OnGetInterception*, *OnSetInterception*).

K dispozícií je deväť variácií adaptácií typu *Interception*. Adaptácie sú aplikované v rôznych krokoch pri vykonávaní metódy alebo prístupu k atribútu. Tieto adaptácie vychádzajú z AspectJ (Kiczales et al., 1997, Kiczales et al., 2001). Keďže adaptácie sú zviazané najmä s telami metód v jazyku JAVA, spájajú sa najmä s triedami a málokedy s rozhraniami, ktoré nedisponujú telami metód. Ďalej v krátkosti opíšeme vybrané adaptácie, pričom pod pojmom pôvodná metóda budeme rozumieť metódu, ku ktorej sa vzťahuje adaptácia. Podrobný popis adaptácií je uvedený v (Quintian, 2004).

Jednou z adaptácií pracujúcich nad metódami je *AroundInterception*, ktorá umožňuje vykonať zdrojový kód spojený s adaptáciou. Najčastejšie sa používa s konštrukciou *if – then – else*, pričom po teste podmienky sa vykonáva pôvodná metóda. Vykonanie pôvodnej metódy spojenej s adaptáciou je vyvolané kľúčovým slovom *proceed*.

Adaptácia *BeforeInterception* (resp. *AfterInterception*) umožňuje vykonanie kódu adaptácie pred (resp. po) volaní pôvodnej metódy.

Adaptácia *OnExceptionInterception* umožňuje odchytenie výnimky a vykonanie kódu adaptácie pred volaním pôvodného spracovania výnimky.

Špeciálnou adaptáciou vzťahujúcou sa k metódam je adaptácia *MethodRedefinition*, ktorá umožňuje voľnú modifikáciu tela metódy, resp. umožňuje predefinovať telo pôvodnej metódy.

Adaptácie vzťahujúce sa k atribútom umožňujú odchytenie udalosti čítania hodnoty atribútu *OnGetInterception* alebo jeho modifikácie *OnSetInterception*.

Fúzia tried A a B je realizovaná formou adaptácie *ClassMerging*, ktorej mechanizmus je inšpirovaný Hyper/J (Ossherand & Tarr, 2000).

Fúzia je zabezpečená nasledovnými spôsobmi:

- rodičovské triedy triedy A sú pridané do zoznamu rodičovských tried triedy B (problém vzniká pri jazykoch neumožňujúcich viacnásobne dedenie),
- do zoznamu metód triedy B sú pridané metódy z triedy A (problém môže nastať pri konflikte v názve, resp. v signatúre metódy),
- každý atribút z triedy A, ktorý neexistuje v triede B, je skopírovaný do zoznamu atribútov triedy B.

Spojenie zoznamov metód tried A a B je realizované cez volanie ďalších adaptácií. V prípade, že sa metóda zo zoznamu metód triedy A nenachádza v triede B, je použitá adaptácia *MethodIntroduction*. V prípade konfliktu názvov, resp. signatúr metód je použitá adaptácia *MethodMerging*.

V prípade adaptácie *MethodMerging* môže dôjsť k viacerým stavom:

- ak sú obe metódy označené ako *abstract*, potom ich fúziou vznikne abstraktná metóda,
- ak metóda A je konkrétna a B abstraktná, potom (adaptácia *MergingWith-Abstract*) výsledkom je konkrétna metóda s telom metódy A,
- ak obe metódy A a B sú konkrétne, je potrebné určiť poradie, v ktorom budú ich telá vykonávané (adaptácie *MergingBefore*, *MergingAfter*).

Na zmenu tela triedy je možné využiť adaptácie pre pridanie metódy *Method-Introduction*, pridanie atribútu *AttributeIntroduction* alebo pridanie rodičovskej triedy *SuperClassIntroduction*.

Každý typ adaptácie obsahuje dve metódy, metódu *Check* pre overenie nastátia udalosti, ktorá kontroluje, či obmedzenie dané konkrétnou adaptáciou nastalo a metódu *Execute* pre vykonanie konkrétnej funkcionality, ktorá opisuje správanie adaptácie.

9.6.3 Príklad kompozície a adaptácie koncernov

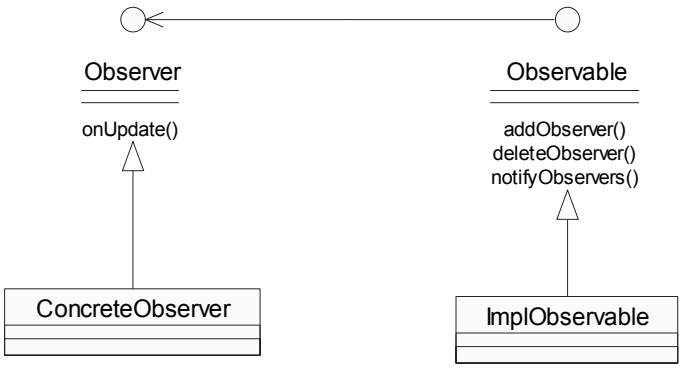
Autori v článku uvádzajú príklad adaptácie jednoduchého grafického rozhrania a návrhového vzoru *Observer*. Problém rozdelili na dva koncerny grafické rozhranie a návrhový vzor *Observer*. Ďalej si popíšeme jednotlivé koncerny.

Koncern návrhový vzor Observer

Návrhový vzor *Observer*, uvedený na obrázku 9-17, pozostáva z klasifikátorov *Observer*, *Observable* a *ImplObservable*, ktoré sú nezávislé od konkrétnej aplikácie. *Observer* a *Observable* korešpondujú s dvomi rolami vo vzore *Observer*. *ImplObservable* je priama implementácia rozhrania *Observable*.

Zo štruktúry vzoru *Observer* vyplýva, že každá trieda implementujúca rozhranie *Observer* musí špecifikovať telo metódy *updateObserver*. Obe rozhrania *Observer* aj *Observable* sú úzko zviazané, pretože každé z nich vyžaduje odkaz na toho druhého pri volaní niektorých z metód.

V príklade 9-7 je uvedená ukážka implementácie návrhového vzoru *Observer*. Potrebné triedy a rozhrania sú umiestnené v balíčku *designpattern.observer*, ktorý predstavuje jeden z koncernov.



Obrázok 9-17. Diagram tried návrhového vzoru Observer podľa (Gamma et al., 1995).

```
01 package designpattern.observer;
02  public interface Observable {
03    public void addObserver(Observer o);
04    public void removeObserver(Observer o);
05    public void notifyObservers();
06   }
07 packagedesignpattern.observer;
08  publicinterfaceObserver{
09    public void updateObserver(Observable o);
10  }
11 packagedesignpattern.observer;
12  import java.util.ArrayList;
13  import java.util.Iterator;
14  import java.util.List;
15  public class ImplObservable implements Observable {
16    protected List observers = new ArrayList();
17    public void addObserver(Observer o){
18      observers.add(o);
19    }
20    public void removeObserver(Observer o){
21      observers.remove(o);
22    }
23    public void notifyObservers() {
24      for(Iterator it=observers.iterator(); it.hasNext();)
25        ((Observer)it.next()).updateObserver(this);
26    }
27  }
```

Príklad 9-7. Ukážka zdrojového kódu návrhového vzoru Observer (Lahire & Quintian, 2006).

Koncern grafického používateľského rozhrania

Druhým z koncernov je grafické používateľské rozhranie, ktoré predstavuje formulár s dvomi tlačidlami (*Click on me!, Exit*) a popiskom. Pri implementácii je využitá knižnica *swing*, konkrétne pre tlačidlá trieda *java.swing.JButton* a pre popisok *java.swing.JLabel*. Pre názornosť je príklad zložený z viacerých tried *Button*, *Label* a *ApplicationInterface*.

Ukážka implementácie je uvedená v príklade 9-8. Grafické rozhranie je implementované v balíčku *application.ibm* a predstavuje koncern.

```
01 package application.ihm;
02   import java.awt.event.ActionEvent;
03   import javax.swing.JButton;
04   public class Button extends JButton{
05     public Button(String text){
06       super(text);
07     }
08     protected void fireActionPerformed(ActionEvent event){
09       super.fireActionPerformed(event);
10     }
11   }
12 package application.ihm;
13   import javax.swing.JLabel;
14   public class Label extends JLabel {
15     public Label(String text){
16       super(text);
17     }
18   }
19 package application.ihm;
20   import java.awt.*;
21   import java.awt.event.*;
22   import javax.swing.*;
23   public class ApplicationInterface extends JFrame {
24     public ApplicationInterface() {
25       super("My application interface");
26       getContentPane().setLayout(new GridLayout(1,3));
27       Button button1 = new Button("Clickonme!");
28       getContentPane().add(button1);
29       Label label1 = new Label("-notext-");
30       getContentPane().add(label1);
31       JButton finish = new JButton("Exit");
32       getContentPane().add(finish);
33       finish.addActionListener(new ExitListener());
34       pack();show();
35     }
36     public static void main(String args[]) {
37       newApplicationInterface();
38     }
39     public class ExitListener implements ActionListener {
40       public void actionPerformed(ActionEvent arg0) {
41         System.exit(0);
42       }
43     }
44   }
```

Príklad 9-8. Ukážka implementácie jednoduchého GUI (Lahire & Quintian, 2006).

Adaptácia koncernu Observer-a na koncern GUI

V príklade 9-9 je uvedená ukážka zdrojového kódu k príkladu koncernov *Observer* a GUI. V prvej časti adaptéra, na riadkoch 4 až 13, sú opísané ciele adaptácie a samotné adaptácie (v prípade *extend class* ide o adaptáciu typu *FeaturesOnlyAdded*, v prípade *extended method* ide o adaptáciu typu *AfterInterception*). Časť druhá, na riadkoch 14 až 20, opisuje ciele adaptácie a samotné adaptácie pracujúce s triedami

vytvorenými v prvej časti (v prípade *inherit* ide o adaptáciu typu *SuperClass-Introduction*, v prípade *introduce method* ide o adaptáciu typu *MethodIntroduction*). V tretej časti adaptéra, na riadkoch 21 až 37, je opísaná potrebná inicializácia pre použitie koncernu *Observer* v kontexte aplikácie.

```
01 package designpattern.observer
02  abstract adapter ObserverAdapter {
03
04    abstract Class target : observableClass
05    abstract Method target : notifyingMethod
06      require notifyingMethod in observableClass.*
07
08    adaptation becomeObservable :
09      extend class ImplObservable with observableClass
10    adaptation notifyingObserver:
11      extend method notifyingMethod() with after
12      { notifyObservers(); }
13 ------------------------------------------------------------
14    abstract Class target
15
16    adaptation becomeObserver :
17      inherit Observer in observerClass
18    abstract adaptation observerUpdate :
19      introduce method public void
20      updateObserver(Observable o) in observerClass
21 ------------------------------------------------------
22    abstract Class target:
23      applicationInitClass
24    abstract Method target:
25      applicationInitMethod
26      require applicationInitMethod in applicationInitClass.*
27    abstract Attribute target:
28      observableInstance
29      require observableInstance in applicationInitMethod.*
30    abstract Attribute target:
31      observerInstance
32      require observerInstance in applicationInitMethod.*
33
34    adaptation initApplication: extend method
35      ApplicationInitClass.applicationInitMethod(...)
36      with after {
37        ObservableInstance.addObserver( ObserverInstance );
38      }
39  }
```

Príklad 9-9. Ukážka zdrojového kódu pre ObserverAdapter (Lahire & Quintian, 2006).

V príklade 9-10 je uvedená konkretizácia adaptéra *ObserverAdapter* pre potreby aplikácie. Ako je uvedené v riadku 2, jedná sa o zloženie koncernov *designpattern.observer* s koncernom *application.ihm*. Pričom podľa riadku 3 ide o rozšírenie adatéra *ObserverAdaptér*. V adaptéry *ApplicationIHM* sú došpecifikované a konkretizované abstraktné ciele adaptácie z hierarchicky nadradeného adaptéra *ObserverAdapter*.

```
01 package application.ihm
02 compose designpattern.observer.* with application.ihm
03 adapter ApplicationIHM extends ObserverAdapter {
04   target observableClass = application.ihm.Button
05   target observerClass = application.ihm.Label
06   target notifyingMethod =
07     application.ihm.Button.fireActionPerformed()
08   target applicationInitClass =
09     application.ihm.ApplicationInterface
10   target applicationInitMethod =
11     applicationInitClass.applicationInterface()
12   target observableInstance = applicationInitMethod.button*
13   target observerInstance = applicationInitMethod.label*
14
15   adaptation observerUpdate :
16   introduce method public void updateObserver(Observable o) {
17     setText(``I have been notified of a button click'');
18   } in observerClass
19 }
```

Príklad 9-10. Konkretizácia adaptéra ObserverAdapter pre potreby príkladu (Lahire & Quintian, 2006).

9.6.4 Realizácia prístupu vo forme prototypu

Opísaný prístup k znovupoužitiu autori implementovali v jazyku JAVA konkrétne ako plugin pre vývojové prostredie Eclipse, pričom JAdaptor, ako nazvali výsledný produkt, pôsobí ako prekompilátor pred JAVA kompilátorom. Za najdôležitejšie úlohy realizované prekompilátorom sa pokladajú:

- získanie informácií k jednotlivým zdrojovým kódom a adaptérom
- kontrola konzistencie informácií, kompozície koncernov
- generovanie zdrojového kódu po realizácii kompozície koncernov

Samotné adaptéry sú opísané formou XML súborov, z ktorých je možné (a potrebné) generovať zdrojový kód adaptérov v jazyku JAVA. Využitím XML sa zabezpečila istá jazyková nezávislosť, pričom pre iný jazyk je potrebné vytvoriť nový generátor zdrojového kódu adaptérov.

9.6.5 Diskusia

Rozdelenie softvéru na koncerny a adaptéry nielenže umožňuje čiastočne prehľadnejšie znovupoužitie častí systému, ale ponúka i akéhosi sprievodcu pri znovupoužití, podľa ktorého môže vývojár postupovať a správne použiť koncern uložený v knižnici. Dá sa povedať, že protokol skladania definuje akúsi metodológiu, akou možno daný koncern znovupoužiť.

Opísaný prístup je pomerne mladý a je spojený s viacerými problémami, napr. pri adaptácií *ClassMerging* môže nastať problém s jazykmi podporujúcimi iba jeden strom dedičnosti, keďže je potrebné spojiť dva zoznamy rodičovských tried, ktoré sa môžu navyše opakovať.

Tento prístup poskytuje veľmi silný nástroj na podporu všeobecného znovupoužitia v súčasnosti používaných objektovo-orientovaných jazykov. Vzhľadom na ponúkané možnosti by bolo vhodné otestovať prístup v kontexte knižnice návrhových vzorov.

Realizácia prototypu formou prekompilátora sa zdá byť výhodnou najmä z hľadiska názornosti spracovania zdrojového kódu, sledovania priebežných výsledkov ako i udržiavania a rozširovania samotného prekompilátora. Otázne je, akým spôsobom je v prototype riešené zobrazenie prípadných syntaktických chýb v zdrojovom kóde pôvodných tried a tiež v zdrojovom kóde adaptérov.

9.7 Zhodnotenie

Východiskom predchádzajúcich analýz boli výsledky výskumu v oblasti návrhových vzorov, konkrétne v oblasti znovupoužitia návrhových vzorov, resp. ich častí. Výsledky analýz a experimentov boli publikované na študentských vedeckých konferenciách IIT.SRC 2005 (článok *Modeling systems using design patterns*) a IIT.SRC 2006 (článok *Comparison of CASE tools based on design patterns source code support*) v Bratislave, na konferenciách s medzinárodnou účasťou OBJEKTY 2005 (článok Izolácia všeobecných častí vzoru Composite) v Ostrave a TVORBA SOFTWARE 2006 (článok Možnosti znovupoužitia častí návrhových vzorov) tiež v Ostrave a na medzinárodnej konferencií JCKBSE 2006 (článok *Reuse of patterns source code*) v Talline.

Postupne sme analyzovali rôzne možnosti znovupoužitia vzorov so zameraním na návrhové vzory. Postupne prechádzame rôzne prístupy od CASE nástrojov cez štandardné i atypické objektovo-orientované jazyky až k rozšíreniam objektovo-orientovaných jazykov a s nimi spojené výhody a nevýhody.

Samotné znovupoužitie návrhových vzorov pomocou CASE nástrojov by sa dalo považovať za úplný základ. CASE nástroje disponujú relatívnou používateľskou prijateľnosťou, i keď proces vkladania nového návrhového vzoru do vlastnej knižnice nie je jednoduchý a pôsobí skôr chaotickým dojmom. Urýchlenie vývoja s využitím CASE nástrojov v podobe jednoduchého a rýchleho vytvárania návrhu s podporou návrhových vzorov prináša so sebou ďalšie skracovanie doby vývoja a s tým súvisiace znižovanie nákladov na vývoj. Sklamanie v podobe nedostatočnej podpory viditeľnosti vzoru a podpory mikroarchitektúr môže byť už onedlho minulosťou, vzhľadom na zainteresovanosť veľkých firiem je vývoj v tejto oblasti pomerne rýchly a zrejme nedostatky v podpore vzorov budú postupne odstraňované.

Znovupoužitie návrhových vzorov s použitím neštandardných jazykov v podobe ohýbania vzoru z knižnice je síce pomerne prehľadné a jednoduché, no na druhej strane len málo projektov môže byť realizovaných v rôznych experimentálnych viac alebo menej známych jazykoch. Atypické jazyky, medzi ktoré v súčasnosti patrí i analyzovaný jazyk Beta, prinášajú rôzne neštandardné prostriedky jednak pre uloženie vzoru, resp. jeho častí do knižnice a tiež pre ohýbanie, modifikovanie a prispôsobovanie zovšeobecneného vzoru pre konkrétnu doménu použitia. I keď autori jazyka JAVA sa pri jeho vývoji v mnohých veciach inšpirovali práve jazykom Beta, prostriedky, ktoré ponúka jazyk JAVA, ani zďaleka nezodpovedajú širokej podpore znovupoužitia vo všeobecnosti v jazyku Beta. Ide zrejme o príliš špecifické prostriedky, ktorých implementácia v nových jazykoch je príliš náročná, a záverečný efekt by nebol postačujúci.

Na druhej strane štandardné objektovo-orientované jazyky ponúkajú iba základné prostriedky pre znovupoužitie. V mnohých prípadoch objektovo-orientované jazyky, ktoré patria do tejto skupiny, disponujú jazykovo špecifickým prostriedkami ako napríklad generické triedy alebo viacnásobné dedenie, ktoré sú síce pomerne jednoducho v danom jazyku použiteľné, no pri nutnosti portovania takéhoto riešenia na iný implementačný jazyk sa vynárajú problémy spojené s realizáciou týchto špecifík pomocou prostriedkov jazyka, ktoré sú k dispozícii. Navyše pri niektorých návrhových vzoroch nemusia postačovať ani tieto jazykovo špecifické prostriedky pre vytvorenie prehľadného riešenia.

Celé snaženie a závery z jednotlivých častí zaoberajúcimi sa znovupoužitím častí návrhových vzorov akoby smerovali k univerzálnym rozšíreniam jazyka, ktoré budú realizovateľné vo väčšej skupine jazykov a aplikačnému programátorovi poskytnú rozšírené možnosti znovupoužitia tried, metód a v konečnom dôsledku i samotných návrhových vzorov. Riešenie sa ponúka formou znovupoužitia koncernov s využitím adaptácií. Analýza riešenia s prekompilátorom pre jazyk JAVA ukázala, že ide o pomerne jazykovo nezávislé riešenie rozširujúce možnosti znovupoužitia zdrojového kódu ako takého. Umožnením vytvárania hierarchií adaptérov a oddelením spoločného zdrojového kódu od špecifického sa v konečnom dôsledku zvýši prehľadnosť riešenia a jeho udržiavateľnosť. Na druhej strane sme nemali možnosť vyskúšať riešenie na vlastnom príklade, a teda nevieme, ako sa prekompilátor chová aj v prípade jednoduchých syntaktických chýb v adaptéroch. Prostriedky, s ktorými prišli autori tohto riešenia, sa zdajú postačujúce na vytvorenie knižnice znovupoužiteľných návrhových vzorov, pričom oddelenie všeobecných častí od doménovo závislých ako i použitie návrhového vzoru z knižnice by malo zostať prehľadné, zrozumiteľné a ľahko udržiavateľné.

Od CASE nástrojov cez atypické a štandardné objektovo-orientované jazyky až k rozšíreniam objektovo-orientovaných jazykov je len vybraná skupina možností ako znovupoužiť zdrojový kód, resp. ako znovupoužiť zdrojový kód návrhového vzoru. Stále zostáva viacero otvorených otázok a alternatívnych možností, ktoré môžu významným spôsobom uľahčiť použitie a znovupoužitie návrhových vzorov.

9.7.1 Otvorené problémy

V zásade sa dá povedať, že sme rozdelili pohľad na znovupoužitie návrhových vzorov na dve základné skupiny, od ktorých sa odvíjajú rôzne problémy. Konkrétne ide o:

- znovupoužitie návrhových vzorov formou CASE nástrojov a
- znovupoužitie návrhových vzorov, resp. ich častí pomocou samotného implementačného jazyka.

Postupne sme identifikovali problémy s viditeľnosťou vzorov v celkovom návrhu, čo je spojené s návrhom a implementáciou prehľadnej a široko použiteľnej notácie a s použitím mikroarchitektúr, ktoré sú ešte i dnes pomerne nejasne definované vo všeobecnosti a tiež chýba definícia, či už formálna alebo semiformálna, skupín spolupracujúcich vzorov vo forme definícií mikroarchitektúr. Zaujímavé sa zdá vytvorenie katalógu mikroarchitektúr ako skupín spolupracujúcich vzorov, pričom štruktúra katalógu by mohla byť podobná štruktúre súčasných katalógov návrhových vzorov.

Pokiaľ hovoríme o jazykoch a ich prostriedkoch, je potrebné analyzovať ďalšie spôsoby znovupoužitia častí návrhových vzorov. Analyzovali sme viacero spôsobov ako znovupoužiť návrhové vzory. Ďalej je potrebné túto skupinu doplniť o možnosti:

- aspektovo-orientovaného programovania,
- použitia anotácií a ďalších neštandardných prostriedkov.

Ďalej je potrebné sledovať najnovšie trendy v oblasti vyvíjajúcich sa verzií existujúcich alebo nových jazykov a s nimi súvisiacu podporu návrhových vzorov priamo v implementačných jazykoch.

Na základe výsledkov analýz rôznych spôsobov znovupoužitia častí návrhových vzorov na úrovni zdrojového kódu by bolo zaujímavé navrhnúť a prototypovať úložisko návrhových vzorov vo forme prístupnej pre priame znovupoužitie.

Zo súčasných analýz vychádza ako jeden z najprehľadnejších spôsobov práve riešenie spojené s rozšírením jazyka o špeciálne konštrukcie, ktoré do istej miery umožňujú neštandardné spôsoby znovupoužitia vo všeobecnosti.

Prípadné porovnanie viacerých spôsobov realizácie úložiska vzorov by prinieslo určite zaujímavé výsledky.

Použitá literatúra

AGERBO, E. – CORNILS, A. (1997). *Theory of language support for design patterns.* Master's thesis, Computer Science Department, Aarhus University, Denmark.

AL-AHMAD, W. (2006). Object-Oriented Design Patterns for Detailed Design. In: *Journal of Object Technology*, Vol. 5, No. 2, March-April 2006, pp. 155-169. URL: http://www.jot.fm/issues/issue_2006_03/article3

ALEXANDER, C. ET AL. (1977). *A pattern language. Towns, buildings, construction.* Oxford University Press, New York, USA, ISBN 0-19-501919-9.

ALEXANDER, C. ET AL. (1979). *The timeless way of building.* Oxford University Press, New York, USA, ISBN 0-19-502402-9.

ALUR, D. ET AL. (2001). *Core J2EE patterns: Best practices and design strategies.* Prentice Hall / Sun Microsystems Press, 1st edition, June 2001, ISBN 0-130-64884-1.

BUSHMAN, F. ET AL. (1996). *A system of patterns.* John Wiley & Sons Ltd., Anglicko, ISBN 0-471-95869-7.

CUNNINGHAM, W. (2006). *Portland pattern repository.* [Online; accessed January 2006] URL: http://c2.com/ppr/index.html

D'ADDERIO, L. – DEWAR, R. – ET AL. (2006). *Systems reengineering patterns.* [Online; accessed May 2006]

URL: http://homepages.inf.ed.ac.uk/perdita/Reengineering/

DIETRICH, J. – ELGAR, C. (2005). A formal description of design patterns using OWL. In: *Proceedings of Australian Software Engineering Conference 2005*, pp. 243-250.

DONG, J. (2003). Representing the applications and compositions of design patterns in UML. In: *Proceedings of the 2003 ACM symposium on applied computing*, ACM Press, New York, USA, pp. 1092-1098, ISBN:1-58113-624-2.

DONG, J. (2002). UML extensions for design pattern compositions. In: *Journal of object technology*, Vol. 1, No. 5, pp. 151-163.

EDEN, A. (2002). A theory of object-oriented design. In: *Information Systems Frontiers,* Vol. 4, Issue 4, December 2002, pp. 379-391, ISSN 1387-3326.

FOWLER, M. (2003). *Patterns of enterprise architecture.* Addison Wesley Professional, Massachusetts, UK, 2003, ISBN 03-211-2742-0.

GAMMA, E. ET AL. (1995). *Design patterns, elements of reusable object-oriented software.* Addison-Wesley Publishing Company, Massachusetts, UK, ISBN 0-201-63361-2.

GEARY D. (2002). A look at the composite design pattern. *Java design patterns.* September, 2002. [Online; accessed May 2005] URL: http://www.javaworld.com/javaworld/jw-09-2002/jw-0913-designpatterns.html

GIL, J. – MAMAN, I. (2005). Micro patterns in java code. In: *Proceedings of the 20th annual ACM SIGPLAN conference on object oriented programming, systems, languages, and applications 2005*, ACM Press, New York, USA, pp. 97-116, ISSN 0362-1340.

JAKUBÍK, J. (2005a). Modeling systems using design patterns. In: *Proceedings of IIT.SRC 2005,* student research conference, Slovak University of Technology, Bratislava, pp. 151-158, ISBN 80-227-2217-0.

JAKUBÍK J. (2005b). Izolácia všeobecných častí vzoru Composite, In: Václav Snášel (ed.) *Objekty 2005: Sborník príspevku desátého ročníku konference,* Fakulta elektrotechniky a informatiky, VŠB – Technická unverzita Ostrava, pp. 74-84, ISBN 80-248-0595-2.

JAKUBÍK, J. (2006). Comparison of CASE tools based on design patterns source code support. In: *Proceedings of IITSRC 2006*, pp. 197-203, ISBN 80-227-2395-9.

KICZALES, G. – LAMPING, J. – ET AL. (1997). Aspect oriented programming. In: *Proceedings of the european conference on object-oriented programming 1997*, Springer-Verlag, London, UK, pp. 220-242.

KICZALES, G. – HILSDALE, E. ET AL. (2001). An overview of aspectj. In: *Proceedings of the european conference on object-oriented programming 2001*, Springer-Verlag, London, UK, pp. 327-353, ISBN 3-540-42206-4.

KIM, D. – FRANCE, R. – ET AL. (2003). A UML-based metamodeling language to specify design patterns. In: *Proceedings of workshop software model engineering with unified modeling language conference 2003*.

LAHIRE, P. – QUINTIAN, L. (2006). New perspective to improve reusability in object–oriented languages. In: *Journal of Object Technology*, Vol. 5., No. 1, January – February 2006.

LAUDER, A. – KENT, S. (1998). Precise visual specification of design patterns. In: *Proceedings of the 12th european conference on object-oriented programming 1998*, pp. 114-134.

MACDONALD, S. ET AL. (2002). Generative Design Patterns. In: *Proceedings of 17th IEEE international conference on automated software engineering 2002*, September 2002, Edinburgh, UK, pp. 23-34.

MADSEN, O. L. – MOLLER – PEDERSEN, B. (1992). Part – objects and their location. In: *Proceeding of international conference on technology of object-oriented languages and systems 1992*, Prentice Hall International, Hertfordshire, UK, pp. 283-297, ISBN 0-13-917436-2.

MAK, J. – CHOY, C. – LUN, D. (2004). Precise modeling of design patterns in UML. In: *Proceedings of the 26th international conference on software engineering 2004*, IEEE Computer Society, Washington, USA, pp. 252-261, ISSN 0270-5257.

OSSHERAND, H. – TARR, P. (2000). Hyper/j: Multi-dimentionnal separation of concern for java. In: *Proceedings of international conference on software engineering 2000*, ACM Press, New York, USA, pp. 734-737, ISBN 1-58113-206-9.

REENSKAUG, T. (1995). *Working with objects: The ooram software engineering method.* Prentice Hall Software, Hertfordshire, UK, ISBN 0134529308.

RIEHLE, D. (2000). *Framework design – a role modeling approach.* Dissertation thesis, Swiss federal institute of technology, Zurich.

TAIBI T. – CHECK LING NGO D. (2003). Formal Specification of Design Patterns – A Balanced Approach. In: *Journal of Object Technology*, Vol. 4, Júl – August 2003, pp. 127-140, URL: http://www.jot.fm/issues/issue_2003_07/article4.

QUINTIAN, L. (2004). *JAdaptor: Un modele pour ameliorer la reutilization des preoccupations dans le paradigme objet.* Ph.D thesis in Computer Science, University of Nice – Sophia Antipolis, France, 2004.

VLISSIDES, J. (1998). Notation, notation, notation. In: *C++ Report*, 1998, . [Online; accessed June 2006]

URL: http://www.research.ibm.com/designpatterns/pubs/ph-apr98.pdf

YACOUB, S. M. - AMMAR, H. H. (2000). Pattern-oriented analysis and design (POAD): A structural composition approach to glue design patterns. In: *Proceedings of the technology of object-oriented languages and systems 2000*, pp. 273, ISBN 0-7695-0774-3.

EDÍCIA VÝSKUMNÝCH TEXTOV
INFORMATIKY A INFORMAČNÝCH TECHNOLÓGIÍ

Mária Bieliková, Pavol Návrat a kol.

ŠTÚDIE VYBRANÝCH TÉM SOFTVÉROVÉHO INŽINIERSTVA (2)

Pokročilé metódy navrhovania programových systémov
Pokročilé metódy získavania, vyhľadávania, reprezentácie
a prezentácie informácií

1. vydanie
Tlač Vydavateľstvo STU v Bratislave

236 strán

2006

ISBN